Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Bancos de Dados Geográficos: Uma Abordagem Orientada a Grafos

Evandro Roberto Mota

Monografia apresentada como requisito parcial
para conclusão do Curso de Computação — Licenciatura

Orientadora
Prof.ª Dr.ª Maristela Terto de Holanda

Brasília
2016

Universidade de Brasília — UnB
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Curso de Computação — Licenciatura

Coordenador: Prof. Dr. Pedro Antônio Dourado Rezende

Banca examinadora composta por:

Prof.ª Dr.ª Maristela Terto de Holanda (Orientadora) — CIC/UnB
Prof.ª Dr.ª Aletéia Patrícia Favacho de Araújo — CIC/UnB
Prof. Dr. Marcio de Carvalho Victorino — CIC/UnB

CIP — Catalogação Internacional na Publicação

Mota, Evandro Roberto.

Bancos de Dados Geográficos: Uma Abordagem Orientada a Grafos / Evandro Roberto Mota. Brasília : UnB, 2016.
145 p. : il. ; 29,5 cm.

Monografia (Graduação) — Universidade de Brasília, Brasília, 2016.

1. NoSQL, 2. Dados Geográficos, 3. Modelo Relacional.

CDU 004.4

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro — Asa Norte
CEP 70910-900
Brasília–DF — Brasil

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Bancos de Dados Geográficos: Uma Abordagem Orientada a Grafos

Evandro Roberto Mota

Monografia apresentada como requisito parcial
para conclusão do Curso de Computação — Licenciatura

Prof.ª Dr.ª Maristela Terto de Holanda (Orientadora)
CIC/UnB

Prof.ª Dr.ª Aletéia Patrícia Favacho de Araújo
CIC/UnB

Prof. Dr. Marcio de Carvalho Victorino
CIC/UnB

Prof. Dr. Pedro Antônio Dourado Rezende
Coordenador do Curso de Computação — Licenciatura

Brasília, 23 de junho de 2016

Dedicatória

Dedico este trabalho à todas as pessoas que direta ou indiretamente contribuíram para minha formação acadêmica e pessoal, possibilitando a conclusão desta importante etapa em minha vida.

Agradecimentos

Agradeço à professora Maristela pela ajuda e orientação durante esses dois semestres de elaboração do trabalho. Ao Victor por me manter motivado. Aos meus irmãos Paulo e Rayane e demais familiares que me acompanharam nessa caminhada. À minha mãe Dionísia pelo suporte, apoio e incentivo durante todos estes anos, provendo educação e valores que me tornaram quem eu sou hoje.

Resumo

Os bancos de dados orientados a grafo, assim como as demais abordagens NoSQL, surgiram como altenativa a problemas específicos que não eram bem atendidos pelo modelo relacional. A grande quantidade de dados gerada atualmente, e a alta escalabilidade horizontal proporcionada pelos NoSQL tem levado a maior adesão dessa abordagem, que agora adapta-se para oferecer suporte aos dados geográficos, cada vez mais presentes em aplicações diversas. Este trabalho tem como objetivo analisar o comportamento de um banco de dados geográficos orientado a grafo em relação à um baseado no modelo relacional, e verificar os desafios do uso desta abordagem orientada a grafo em questões de armazenamento e manipulação de dados geográficos.

Palavras-chave: NoSQL, Dados Geográficos, Modelo Relacional.

Abstract

Graph databases, like other NoSQL approaches, came as an alternative to solve specific problems that weren't well fit for the relational model. The huge amounts of data produced, and the high capability for horizontal scalability provided by NoSQL databases has led to the increasing adoption of these systems, that now adapt to support geographic data, each day more present in diverse applications. This work aims to analyse the behavior of a geographic graph database in comparison with one based in the relational model, and to check the challenges of the use of this graph oriented approach regarding the storage and management of geographic data.

Keywords: NoSQL, Geographic Data, Relational Model.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Dados geográficos, ou geograficamente referenciados, são aqueles que descrevem algum fenômeno geográfico em que sua localização esteja associada a uma posição sob a superfície terrestre [53]. Estes dados são caracterizados por quatro aspectos: a descrição do fenômeno; sua posição geográfica; seus relacionamentos espaciais com outros fenônemos; e o instante ou intervalo de tempo em que o fenômeno é válido.

O crescente interesse pela manipulação de dados geográficos para representar aspectos urbanos, ambientais e socioeconômicos através do computador impulsionou a evolução dos Sistemas de Informação Geográfica (SIG) [56], capazes de capturar, modelar, manipular, recuperar, analisar e apresentar dados geograficamente referenciados [84].

Por muitos anos, os bancos de dados relacionais foram solução para inúmeros problemas envolvendo o gerenciamento de dados, inclusive os geográficos. Ainda hoje esses sistemas são utilizados na maior parte dos casos. No entanto, existem algumas situações em que os bancos de dados relacionais podem não apresentar a eficiência desejada.

Com a forte difusão da Internet, a quantidade de dados gerada tem sido tão grande que apenas este fator já apresentaria um desafio em questão de escalabilidade aos bancos relacionais. Entretanto, estes dados têm se tornado mais complexos e menos estruturados, dificultando também seu armazenamento no esquema rígido do modelo relacional [61].

O surgimento dos bancos de dados não relacionais, conhecidos como NoSQL (*Not Only SQL* ou Não Apenas SQL, em tradução livre), se apresentou como uma alternativa aos Sistemas Gerenciadores de Bancos de Dados (SGBD) relacionais ao possibilitarem o desenvolvimento de aplicações com alta escalabilidade horizontal e capacidade de gerenciar dados em larga escala [59]. Esses bancos de dados NoSQL são classificados de acordo com a estrutura de dados utilizada, dentre elas destacam-se as abordagens orientadas a colunas, chave/valor, documentos e grafos [25].

Os bancos de dados em grafo destacam-se por armazenar os relacionamentos de forma explícita no banco de dados, conferindo-lhes o mesmo nível de expressividade dos demais registros, e são otimizados para situações em que a detecção e o reconhecimento de padrões sejam essenciais.

A crescente demanda por dados geográficos provocou a necessidade de sistemas de bancos de dados capazes também de gerenciar e analisar dados geoespaciais. Funcionalidade já presente nos principais bancos de dados relacionais, os emergentes NoSQL agora se adaptam para fornecer suporte a dados geográficos, mantendo-se como uma alternativa ao modelo relacional.

1.1 Objetivos

Este trabalho tem como objetivo geral a realização de uma análise comparativa entre um banco de dados baseado em grafo e um banco de dados relacional, no contexto de armazenamento e manipulação de dados geográficos, tendo como premissa a utilização de ferramentas livres e de código aberto.

Para que o objetivo geral seja atingido, este trabalho tem como objetivos específicos:

- Apurar o suporte de ferramentas NoSQL ao armazenamento de dados geográficos;
- Implementar os bancos de dados geográficos baseado em grafo e relacional contendo o mesmo conjunto de dados;
- Executar consultas espaciais equivalentes em ambos os bancos de dados;
- Comparar os resultados obtidos pelas consultas;
- Identificar vantagens e desvantagens da utilização de banco de dados baseado em grafo para armazenamento de dados geográficos.

1.2 Metodologia

Este trabalho foi desenvolvido em duas partes, sendo a primeira delas teórica e a segunda, prática.

Na primeira parte foi realizada uma revisão bibliográfica utilizando livros e artigos sobre o tema de banco de dados, mais especificamente, em bancos de dados geográficos e NoSQL. A segunda parte foi destinada à manipulação de dados geográficos através de um banco de dados baseado em grafo, por meio da implementação de um estudo de caso que serviria de base para as análises posteriores.

A Figura 1.1 representa um diagrama da metodologia adotada para a elaboração do trabalho, em que inicialmente foi realizado o estudo bibliográfico que serviu como base para definição dos SIG, SGBD e NoSQL. Em seguida, foram implementados os bancos de dados relacional e em grafo utilizando dados geográficos, onde foram executadas consultas espaciais equivalentes de modo a realizar uma análise comparativa e chegar às conclusões finais.

Figura 1.1: Esquema da Metodologia Adotada para a Elaboração do Trabalho.

1.3 Estrutura do Trabalho

Além deste capítulo, este trabalho está estruturado nos demais capítulos:

- Capítulo 2: Introduz os SIG, abordando os conceitos básicos relacionados, sua evolução, estrutura e principais funcionalidades. Apresenta ainda os bancos de dados e a evolução desses sistemas para oferecer suporte aos dados geográficos, originando o surgimento dos Bancos de Dados Geográficos.
- Capítulo 3: Apresenta uma visão geral a respeito dos bancos de dados NoSQL, contextualizando e abordando os aspectos fundamentais e as principais características desses sistemas.
- Capítulo 4: Aborda as características desejáveis em um banco de dados geográficos não relacional, e apresenta funcionalidades de sistemas NoSQL que oferecem suporte a dados geográficos.
- Capítulo 5: Contém a análise comparativa entre um banco de dados baseado em grafo e um banco de dados relacional utilizando dados geográficos a partir de critérios como a inserção, o armazenamento e a visualização de dados, o desempenho das consultas espaciais e o suporte e usabilidade do sistema.
- Capítulo 6: Expõe as conclusões obtidas a partir do desenvolvimento deste trabalho e apresenta algumas sugestões para trabalhos futuros.

Capítulo 2

Bancos de Dados Geográficos

Este capítulo introduz na Seção 2.1 os conceitos de dado, informação e conhecimento. A Seção 2.2 aborda os Sistemas de Informação Geográfica, apresentando sua evolução, seus possíveis campos de aplicação, sua estrutura e componentes. As principais funcionalidades e características dos Bancos de Dados são apontadas na Seção 2.2.4, e dos Bancos de Dados Geográficos na Seção 2.3.

2.1 Informação Geográfica

Dados consistem de números, textos ou símbolos livres de contexto. São fatos brutos, úteis ou não, e desprovidos de qualquer significado próprio além de sua existência [17]. O termo informação representa dados que foram organizados e selecionados para a obtenção de significado servindo a algum propósito específico [44]. Informação é extraída a partir dos dados por meio de processamento ou interpretação dentro de um contexto.

Conhecimento é uma abstração pessoal [75], agregando valor à informação por meio de interpretação baseada em um contexto, experiência ou propósito particular. O modo que a informação é interpretada determina o conhecimento, que será diferente para diferentes pessoas, ainda que se tratando do mesmo conjunto de informações [44].

A Figura 2.1 ilustra a pirâmide hierárquica formada por estes conceitos. Os dados servem de base para a informação, que por sua vez, sustenta todo o conhecimento. Não há informação sem dados, assim como não é possível obter conhecimento ausente de informação [49].

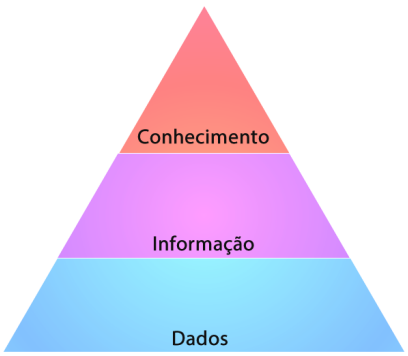


Figura 2.1: A Pirâmide do Conhecimento, adaptada de [49].

Dados geográficos são dados utilizados para descrever alguma entidade de acordo com sua localização no espaço tridimensional dentro de um sistema de coordenadas, podendo ser armazenados e processados de acordo com a informação que se pretende obter a partir deles [51].

Informação geográfica consiste no processamento de dados geográficos. É composta por duas características principais: a informação propriamente dita — contendo nome, descrição, valor, dentre outros atributos — e sua localização espacial, ou seja, sua posição no espaço geográfico [30]. Ela permite aplicar princípios gerais às localidades específicas, monitorar condições e acontecimentos, e diferenciar um lugar de outro [18].

2.2 Sistemas de Informação Geográfica

SIG é um sistema de informação computacional capaz de capturar, modelar, manipular, recuperar, analisar e apresentar dados geograficamente referenciados [84]. Dados geograficamente referenciados são aqueles que descrevem algum fenômeno geográfico em que sua localização esteja associada a uma posição sob a superfície terrestre.

Segundo Lisboa [53], quatro aspectos caracterizam um dado georreferenciado:

- a descrição do fenômeno geográfico;
- sua posição geográfica;
- relacionamentos espaciais com outros fenômenos geográficos;
- instante ou intervalo de tempo em que o fenômeno é válido.

Os SIG tem sido utilizados desde o final da década de 1960 [28], mas foi na década de 1980 em que o crescente interesse pela manipulação de dados geográficos para representar aspectos urbanos, ambientais e socioeconômicos, através do computador, impulsionou a evolução destes sistemas [56].

A rápida expansão, o desenvolvimento simultâneo ao redor do mundo, e o campo de aplicações diversificado acarretaram em dificuldades na definição dos SIG [55]. Na literatura é possível encontrar diferentes definições para estes sistemas, dentre essas destacam-se:

"*Um SIG é definido como qualquer conjunto de procedimentos, manuais ou computacionais, utilizados para armazenamento e manipulação de dados geograficamente referenciados*" (Aronoff, 1989 [16]).

"*Caso especial de sistema de informação, em que sua base de dados consiste de observações sobre características, atividades, ou eventos distribuídos no espaço, podendo ser definido por pontos, linhas ou áreas*" (Dueker, 1987 [35]).

"*Poderoso conjunto de ferramentas para coletar, armazenar, recuperar, transformar e visualizar dados espaciais do mundo real*" (Burrough, 1986 [22]).

"*Sistema de apoio à decisões que envolve a integração de dados espacialmente referenciados em um ambiente de resolução de problemas*" (Cowen, 1988 [29]).

Atualmente, os SIG são reconhecidos pela composição de três premissas essenciais [56]:

- **Sistema**: um ambiente automatizado e integrado com procedimentos para entrada, armazenamento, manipulação e saída de dados geográficos;
- **Informação**: tudo aquilo que se possa extrair algum significado por meio de perguntas ao banco de dados e obtido através da organização dos dados espaciais em um modelo do mundo real;
- **Geográfica**: apesar de ser possível que um SIG também gerencie dados não geográficos, seu enfoque está nos objetos espaciais e sua localização, medida por escalas geográficas baseadas em sistemas de coordenadas e referenciamento.

2.2.1 A Evolução dos SIG

O desenvolvimento dos SIG não ocorreu de forma isolada de outras tecnologias. Analisar a relação entre um sistema de informação geográfica e sistemas de desenho assistido por computador — *Computer Aided Design* (CAD), de cartografia computacional, de gerenciamento de banco de dados e de sensoriamento remoto é importante para o entendimento de um SIG, uma vez que este é um subconjunto dos sistemas anteriores [55] como mostra a Figura 2.2:

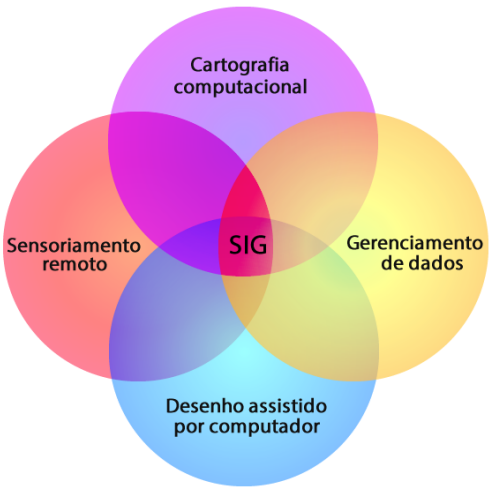


Figura 2.2: A Relação entre SIG e Sistemas de Desenho Assistido por Computador, de Cartografia Computacional, de Gerenciamento de Dados e de Sensoriamento Remoto, adaptado de [55].

- Sistemas CAD são baseados em gráficos e utilizam símbolos como primitivas para o desenho e o projeto de novos objetos. Utilizam-se de relações topológicas simples e em geral lidam com um volume de dados relativamente pequeno, além de não possuírem funções de análise mais complexas.
- Sistemas de cartografia computacional focam na exibição dos dados, ao invés de seu armazenamento ou análise. Utilizam estruturas de dados simples e que não contém informações topológicas. Podem estar associados a um sistema de banco de dados

para armazenamento e funções simples de recuperação de dados. Sua ênfase está na produção de mapas de alta qualidade.

- Sistemas gerenciadores de bancos de dados são sistemas desenvolvidos para fins de armazenamento e recuperação de dados. É um dos principais elementos de um sistema *web*, mantendo a informação persistente a uma aplicação. Contudo, no período em que os SIG começaram a emergir, apresentavam certa limitação na implementação de funções para análise espacial.
- Sistemas de sensoriamento remoto trabalham com a manipulação, o armazenamento, a exibição e a extração de informações sobre objetos, fenômenos ou localidades, por meio de dados advindos de dispositivos que não estejam em contato direto com o objeto em estudo, geralmente, providos por satélite.

Os primeiros SIG careciam de funções com o objetivo de analisar dados espaciais, visto que foram inicialmente desenvolvidos apenas como ferramentas para o armazenamento e recuperação de informação geográfica [38]. Além da capacidade de análise dos dados espaciais ter se tornado o elemento chave dos SIG, esta habilidade é a principal característica que os distingue de sistemas cujo objetivo é simplesmente a geração de mapas [42].

2.2.2 Estrutura de um SIG

Os elementos que compõem um SIG estão representados na Figura 2.3, de acordo com Casanova et al. [24], que descrevem sua estruturação da seguinte forma: no nível mais acima, uma interface com o usuário define como o sistema é operado e controlado; no nível intermediário, composto por três módulos, encontram-se os mecanismos de processamento de dados espaciais, oferecendo ferramentas para entrada e saída de dados, funções de consulta e análise espacial, e visualização e plotagem de mapas. Por fim, no nível mais baixo do sistema, os SIG oferecem armazenamento e recuperação de dados através de um sistema gerenciador de dados geográficos. A Figura 2.3 indica ainda como estes componentes de um SIG se relacionam:

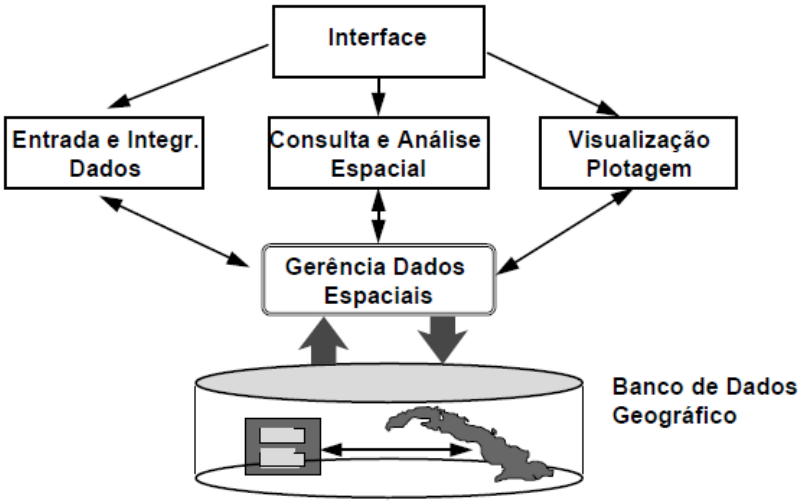


Figura 2.3: Estrutura Geral de Sistemas de Informação Geográfica [24].

2.2.3 As Aplicações de um SIG

Por terem incorporado funcionalidades de outros sistemas, principalmente, de cartografia computacional e de processamento de imagens, grande parte dos estudos envolvendo Sistemas de Informações Geográficas se concentram nessas áreas. Entretanto, os SIG abrangem uma variada gama de aplicações, e algumas delas sequer se encontram no domínio da geografia.

Em seu livro introdutório sobre Sistemas de Informações Geográfcas, Kennedy [51] lista doze aplicações mais comuns a um SIG. Dentre elas destacam-se:

- **Meio Ambiente**: informações a respeito de determinada área do meio ambiente são importantes, principalmente, para verificar seu estado de preservação, além de auxiliar em medidas de conservação, combate ao desmatamento, identificar situações de impacto ambiental, e possíveis melhorias ao ecossistema em questão. É possível ainda manipular dados referentes a clima, solo, hidrografia, superfície geológica, fauna e flora.
- **Energia**: dados geográficos em recursos energéticos podem contribuir para maior conscientização da sua utilização, fornecendo análises em: disperdício, poluição e disponibilidade de recursos.
- **Demografia**: um SIG com dados de uma população pode prever padrões e utilizá-los em prol da comunidade, auxiliando na melhor distribuição de serviços públicos como saúde e educação, por exemplo. O armazenamento de dados referentes a idade, localização, fecundidade e consumo, dentre outras variáveis, possibilita ainda a realização de análises demográficas em relação à natalidade, aos movimentos migratórios e ao crescimento populacional.
- **Prevenção de crimes**: o sistema criminal pode se beneficiar de um SIG na detecção de áreas mais propensas às atividades criminosas, e fornecendo informações necessárias para melhor remanejamento de reforços policiais, alocação tática de patrulhas, planos de combate mais eficazes e instalação de postos policiais em locais estratégicos.
- **Comunicação**: cada vez mais presente em nossas vidas, os sistemas de comunicação demandam uma grande infraestrutura, que se alocada de forma incorreta pode causar danos ao meio ambiente e oferecer risco à população. O armazenamento de dados geográficos voltados para essa área pode ser um reforço na instalação de: linhas de transmissão, torres de comunicação e antenas de telefonia.
- **Transporte**: análise de dados espaciais é fundamental para um sistema de transporte eficiente. Não apenas em nível local, mas nacional e global. Assim, um SIG tem muito a contribuir em relação à movimentação de pessoas nos âmbitos sociais e econômicos, não se restringindo apenas ao transporte terrestre, mas abrangindo também portos, aeroportos, ferrovias, tráfego comercial e turismo.

2.2.4 Bancos de Dados

Os dados são o cerne dos SIG [84] e base para todas as consultas, análises e tomadas de decisão proporcionadas. É necessário, portanto, que seu banco de dados seja bem estruturado e implementado, caso contrário, a qualidade de tais análises é comprometida [39].

Um banco de dados é comumente descrito como uma coleção de dados relacionados. No entanto, Elmasri et al. em [36] consideram que essa definição é muito genérica e dá margem para interpretações incorretas do conceito, e definem que um banco de dados, além de ser uma coleção de dados relacionados, deve possuir as seguintes propriedades implícitas:

- representa aspectos do mundo real;
- é uma coleção lógica de dados com algum significado inerente. Uma organização de dados aleatórios não pode ser considerada um banco de dados;
- é projetado, construido e povoado por dados que atendem a uma proposta específica;
- possui um grupo de usuários definido e aplicações preconcebidas de acordo com o interesse deste grupo.

Os bancos de dados são administrados por programas computacionais chamados Sistemas Gerenciadores de Bancos de Dados (SGBD). Um sistema de banco de dados tem como objetivo o armazenamento de dados, permitindo que o usuário gerencie esses dados quando necessário [31]. Um SGBD envolve quatro componentes principais: dados, hardware, software e usuários, como ilustra o esquema simplificado na Figura 2.4.

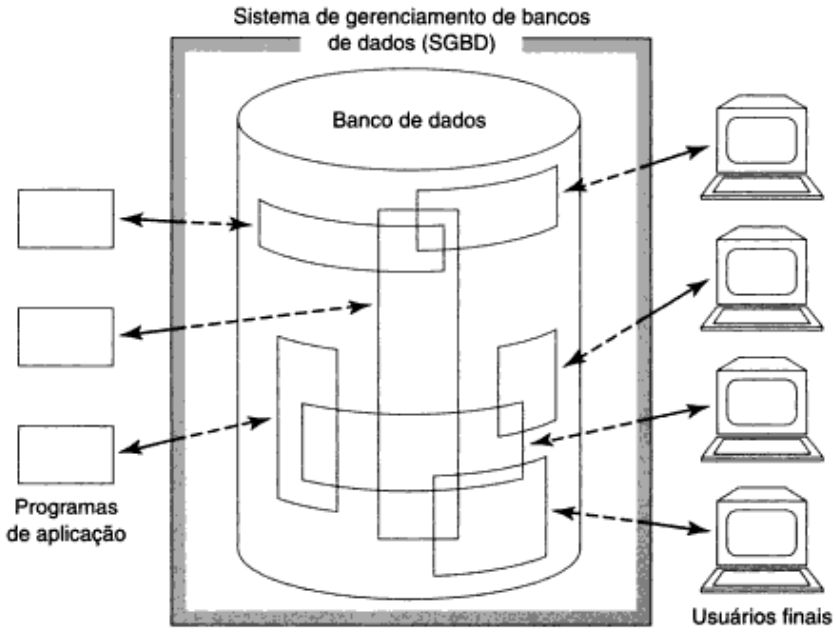


Figura 2.4: Representação Simplificada de um Sistema de Banco de Dados [31].

O *hardware*, representado na Figura 2.4 pelo banco de dados, é o dispositivo de armazenamento físico de dados. Entre o banco de dados e os usuários do sistema existe uma camada de software responsável por realizar interface com o usuário, isolando-o de detalhes em nível de *hardware*. Os usuários são representados pelos programas de aplicação — que permitem que usuários destes programas acessem o banco de dados de forma interativa — e os usuários finais, que além de acessar o banco através de programas de aplicação, podem fazê-lo diretamente pelo SGBD.

Os SGBD substituem os tradicionais sistemas de arquivos [36], no qual cada usuário deve definir e implementar os arquivos necessários para uma aplicação específica como parte da própria aplicação, além disso, ainda que compartilhem da mesma coleção de dados, suas informações são mantidas em arquivos diferentes, pois um usuário pode precisar de dados que não necessariamente estarão contidos nos arquivos de outro. Tal abordagem provoca redundância de dados, um considerável espaço de armazenamento desperdiçado, e um esforço desnecessário para manter dados comuns atualizados.

Os SGBD são ferramentas para a criação e a gerência de grandes quantidades de dados de forma eficiente e persistente [40], isolando o usuário de detalhes em nível de *hardware*.

A Tabela 2.1 ilustra as principais diferenças entre os Sistemas Gerenciadores de Bancos de Dados e os tradicionais Sistemas de Arquivos. Como pode ser observado, em contraste ao SGBD, o sistema de arquivos apresenta redundância e dependência dados-aplicação; e qualidades como interface amigável, segurança e integridade dependem da implementação utilizada, enquanto os SGBD proveem tais características de forma nativa [36].

Tabela 2.1: Comparação entre os SGBD e os Sistemas de Arquivos.

SGBD	**Sistema de Arquivos**
Armazena os metadados	Definição é parte da aplicação
Otimizado para reduzir redundâncias	Há bastante redundância
Independência dados-programa	Dependência dados-programa
Eficiência, concorrência, segurança, integridade	Variável de acordo com a aplicação
Interface amigável	Interface depende da implementação

As principais vantagens garantidas pela utilização dos sistemas gerenciadores de bancos de dados, de acordo com Ramakrishnan [69], são:

- **Independência de dados**: o SGBD provê uma visão abstrata dos dados, isolando a aplicação de detalhes sobre armazenamento e representação de dados e, consequentemente, tornando a aplicação independente de tais particularidades;
- **Integridade e segurança de dados**: o SGBD permite a imposição de regras aos dados armazenados, e controle de acesso aos dados de acordo com a autenticação do usuário;
- **Administração de dados**: quando dados são compartilhados entre diversos usuários, centralizar a administração destes dados pode representar melhorias significativas, minimizando a redundância, e tornando a recuperação de dados mais eficiente;
- **Transações concorrentes e recuperação a falhas**: o gerenciamento transacional é realizado de tal forma que cada transação seja realizada de forma isolada em

relação às outras transações acontecendo simultaneamente. Além disso, o SGBD possui mecanismos de *backup* e de recuperação de dados em caso de falhas, erros, ou mal uso do banco de dados;

- **Redução no tempo de desenvolvimento de aplicações**: todas as funcionalidades anteriormente descritas, juntamente com uma interface de alto nível, favorecem o desenvolvimento de aplicações mais robustas em menor tempo.

2.3 Bancos de Dados Geográficos

Se por um lado, os sistemas de informações geográficas evoluíram incorporando características de sistemas gerenciadores de bancos de dados, como visto na Seção 2.2, por outro lado, os SGBD também evoluíram para oferecer suporte aos dados geográficos de forma integrada.

Bancos de dados geográficos são sistemas que oferecem funcionalidades adicionais em relação aos bancos de dados tradicionais para o suporte aos tipos de objeto espacial em sua modelagem, permitindo o armazenamento de representações de fenômenos geográficos do mundo real para serem utilizados por sistemas de informação geográfica [46].

Um banco de dados geográfico atenta-se às premissas listadas, anteriormente, para bancos de dados em geral como armazenamento, integridade, concorrência, e consultas — especialmente de dados geográficos, mas não se restringindo a eles.

Um SIG, em contrapartida, tem seu enfoque dado em operações de análise espacial, adquirindo informações mais aprofundadas a partir dos dados geográficos, como distância, elevação, topologia, etc.

É comum, portanto, a integração das duas tecnologias em uma única aplicação. O SIG se encarrega da parte de análise, enquanto a área de banco de dados fica por conta dos SGBD com suporte a dados geográficos [73].

2.3.1 Conceitos Básicos

O *US National Digital Cartographic Standard*, padrão nacional de cartografia digital dos Estados Unidos, definiu em [43] os conceitos fundamentais específicos da área de banco de dados geográficos. Alguns destes conceitos utilizados neste trabalho, são:

- **Identidade**: elementos do mundo real modelados em um banco de dados geográfico têm duas identidades: que são o próprio elemento da realidade, denominado entidade; e a representação do elemento no banco de dados, denominado objeto;
- **Entidade**: é qualquer fenômeno geográfico sob o domínio de interesse;
- **Objeto**: é a representação digital de toda (ou parte) da entidade, podendo variar de acordo com a escala utilizada;
- **Tipo de Entidade**: é a descrição e o agrupamento de fenômenos similares que são representados e armazenados de maneira uniforme;
- **Tipo de Objeto Espacial**: a representação digital dos tipos de entidade em um banco de dados geográfico requer a seleção do tipo adequado de objeto espacial de acordo com suas dimensões. Os tipos de objeto espacial são classificados em:

  - 0D: ponto - um objeto com posição no espaço, mas sem comprimento;
  - 1D: linha - um objeto com comprimento, composto por pelo menos dois objetos 0D;
  - 2D: área - um objeto com comprimento e largura, composto por pelo menos três objetos 1D;
  - 3D: volume - um objeto com comprimento, largura e altura, composto por pelo menos quatro objetos 2D.

- **Atributo**: descreve as características das entidades representadas geralmente de forma não espacial, como nome do município, estado a que pertence e número de habitantes, por exemplo;
- **Camada**: objetos espaciais podem ser agrupados em camadas, geralmente contendo um único tipo ou grupo de entidades relacionadas.

2.3.2 Armazenamento

Embora existam diversas formas nas quais os dados geográficos podem ser armazenados, Lisboa em [53] aponta duas vertentes com maior destaque na representação dos componentes espaciais associados às informações geográficas: a abordagem matricial e a abordagem vetorial.

Abordagem Matricial

No modelo matricial, também conhecido como *raster*, o espaço geográfico a ser representado é dividido em uma grade regular definida por uma matriz $M(i, j)$, composta por i colunas e j linhas que delimitam as células, também denominadas de *pixels (picture elements)* ou elementos de imagem, onde cada célula é referenciada por uma coordenada indicando sua posição na matriz a partir da linha e coluna em que se encontra [24].

Quando a informação é representada no modelo matricial, os detalhes e as variações dentro de uma mesma célula são perdidos, e a ela é geralmente atribuído um único valor [44]. A definição desse valor é dada em geral de acordo com a porção mais predominante dentro da célula.

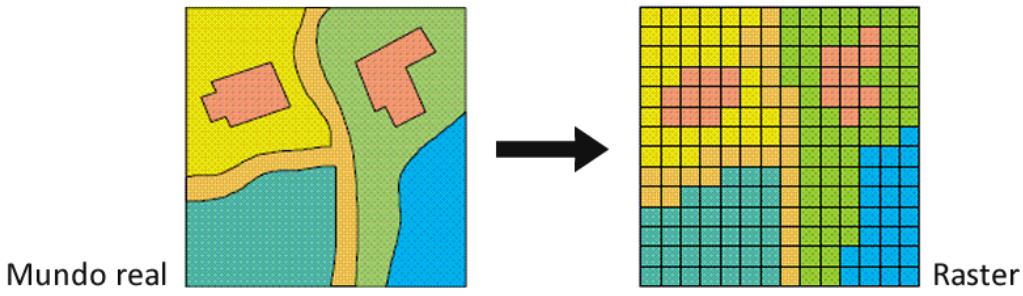


Figura 2.5: Transformação de Imagem Real para Formato *Raster*, adaptado de [3].

A Figura 2.5 ilustra o resultado de um armazenamento de dados no formato *raster*, em que a representação do mundo real é dividida em uma grade de células retangulares que tem seu valor definido pela característica principal nelas contidas.

A resolução de uma imagem matricial corresponde à dimensão da menor unidade do espaço considerado [53]. Quanto menor a dimensão das células, maior a resolução da imagem matricial, como pode ser observado na Figura 2.6, que ilustra uma imagem sendo armazenada em matrizes com dimensões de células diferentes. Quanto menor a dimensão destas células, mais próxima à original será a representação *raster*, e quanto maior a dimensão, maior a perca de detalhes, fazendo com que a representação *raster* se assemelhe menos com a imagem original.

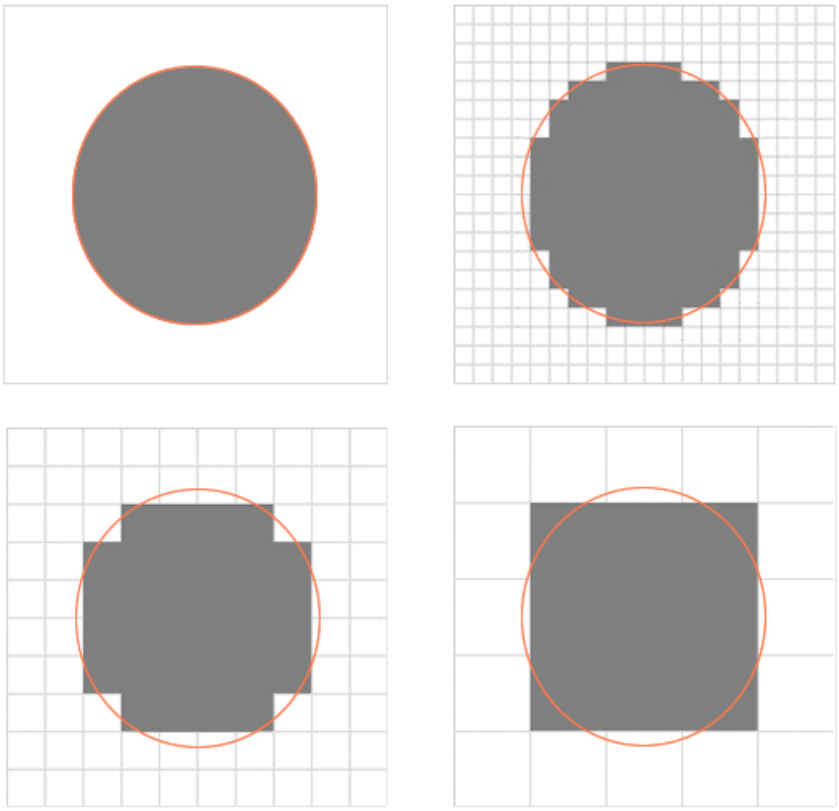


Figura 2.6: Relação entre Qualidade da Imagem *Raster* e o Tamanho das Células.

O tamanho de armazenamento no modelo matricial também está diretamente relacionado com o tamanho dos *pixels*, pois quanto menor os *pixels*, mais valores precisarão ser armazenados no banco de dados. Deve haver, portanto, um balenceamento entre nitidez de imagem e espaço de armazenamento, de acordo com a finalidade da aplicação [63].

Uma vantagem deste modelo é que a representação matricial é tratada de forma natural por computadores devido ao suporte a manipulação de vetores e matrizes provido pelas principais linguagens de programação [84].

Abordagem Vetorial

No modelo vetorial, as entidades do mundo real são representadas pelos três construtores básicos: ponto, linha e polígono, exemplificados na Figura 2.7, e descritos por Câmara [23] como:

- O ponto corresponde a um par ordenado (x, y) de coordenadas espaciais e pode ser utilizado para a localização de entidades ou ocorrências no espaço.
- A linha é formada por uma lista de coordenadas de pontos conectados e representa entidades que possuem feições unidimensionais.
- Um polígono é estruturado como uma cadeia fechada de seguimentos de linha, representando entidades bidimensionais através da delimitação de seu perímetro.

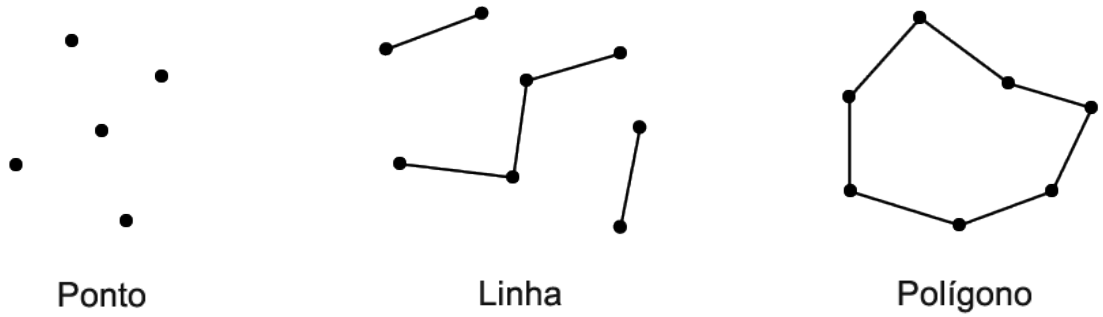


Figura 2.7: Construtores Básicos: Ponto, Linha e Polígono.

A representação vetorial é mais eficiente em termos de armazenamento em relação ao *raster*, pois nem todas as posições do espaço precisam ser referenciadas, e apenas os pontos de interesse são armazenados [84].

As principais diferenças entre os modelos matricial e vetorial podem ser observadas na Tabela 2.2. Como pode ser observado, o modelo matricial possui estrutura mais simples e fornece um tratamento mais adequado às imagens obtidas por satélite, em contrapartida, o modelo vetorial tem estrutura mais compacta, propicia melhor armazenamento topológico e realiza cálculos espaciais com maior precisão [44].

Tabela 2.2: Comparação entre os Modelos de Representação de Dados Geográficos Matricial e Vetorial.

Modelo Matricial	**Modelo Vetorial**
Estrutura mais simples	Estrutura mais compacta
Melhor representação de alta variação espacial	Saídas gráficas de melhor qualidade
Tratamento adequado de imagens por satélite	Cálculos de medidas espaciais mais precisos
Difícil representação de relacionamentos	Melhor armazenamento topológico

2.4 Conclusão

Neste capítulo foram introduzidos os conceitos de dado e informação geográfica, para em seguida serem apresentados os Sistemas de Informação Geográfica, abordando suas diferentes definições, áreas de aplicação, evolução ao longo dos anos, e a estrutura interna destes sistemas.

O banco de dados apresenta grande importância dentre os componentes desta estrutura, pois é nele que estão contidos os dados que servem de base para todas as consultas, análises e tomadas de decisão que os SIG proporcionam.

Devendo ser bem modelado e implementado a fim de atribuir maior qualidade e confiabilidade à aplicação, o banco de dados é administrado pelos Sistemas Gerenciadores de Bancos de Dados — SGBD, que dada a crescente demanda, evoluíram para oferecer suporte aos dados geográficos de forma integrada.

Bancos de dados geográficos são sistemas que oferecem funcionalidades adicionais em relação aos bancos de dados tradicionais para o suporte aos tipos de objeto espacial em sua modelagem, permitindo o armazenamento de representações de fenômenos geográficos do mundo real para serem utilizados por sistemas de informação geográfica. Apesar das diversas alternativas de modelagem, a maioria dos bancos de dados geográficos são baseados no modelo relacional [63], e como tal, apresentam limitações de escalabilidade.

Os bancos de dados não relacionais, que surgiram como uma alternativa de solução para essa e outras limitações do esquema rígido do modelo relacional, são abordados no próximo capítulo.

Capítulo 3

Bancos de Dados NoSQL

Este capítulo apresenta uma visão geral sobre os bancos de dados não relacionais. Inicialmente, é feita uma contextualização, e em seguida as propriedades e características destes bancos são apresentadas. Por fim, a Seção 3.3 aponta os principais tipos e categorias de bancos de dados não relacionais, explicitando cada um deles.

3.1 Contextualização

A expansão da Internet e, principalmente, das redes sociais nos últimos anos trouxeram a necessidade de gerenciamento de grandes volumes de dados, o chamado *Big Data* [57]. A quantidade de dados gerados tem crescido rapidamente de forma exponencial, e seu formato se tornou mais complexo, menos estruturado, mais dinâmico e flexível, dificultanto ou até mesmo impossibilitando seu armazenamento em tabelas [61], como no modelo relacional.

Os bancos de dados não relacionais, também conhecidos como NoSQL evoluiram quando grandes empresas como a *Amazon* e o *Google* decidiram ir além do que o modelo relacional lhes propiciava, desenvolvendo o DynamoDB [34] e o BigTable [26], abordagens orientadas a chave/valor e colunas, respectivamente, para tratar com dados em grande escala e obter alta escalabilidade horizontal.

3.2 Características

Apesar de existirem há algum tempo e serem cada vez mais frequentes em aplicações da *web*, não há uma definição amplamente aceita do que sejam propriamente os sistemas NoSQL. Entretanto, existem seis funcionalidades chaves que ajudam a identificá-los [25]:

- Escalabilidade horizontal e processamento distribuído;
- Replicação e particionamento;
- Interface simples;
- Transações, geralmente, não ACID (Atomicidade, Consistência, Isolamento e Durabilidade);
- Uso eficiente de índices distribuídos;

- Adicionar novos atributos dinamicamente aos registros.

Além disso, a maioria dos bancos de dados NoSQL utilizam uma linguagem de consulta diferente do SQL (*Structured Query Language*), não possuem um modelo de dados com esquema definido e são de código aberto [52].

3.2.1 O Teorema CAP

No ano 2000, Eric Brewer formulou o Teorema CAP [41], que é até hoje amplamente utilizado por grandes empresas e pela comunidade NoSQL em geral [76]. O teorema define que para qualquer banco de dados distribuído existem três propriedades interdependentes básicas. São elas:

- **Consistência** (*Consistency*): os dados devem permanecer consistentes após cada operação, garantindo que todos os usuários possam ler do banco a versão mais atualizada dos dados;
- **Disponibilidade** (*Availability*): a garantia de que toda operação, sucedida ou não, termine com uma resposta intencionada. Alcançada por meio de vários servidores agindo como uma base única através do compartilhamento de dados e replicação;
- **Tolerância a Particionamento** (*Partitioning Tolerance*): o sistema de banco de dados deve permanecer disponível ainda que partes do mesmo estejam inativas devido a algum problema ou manutenção.

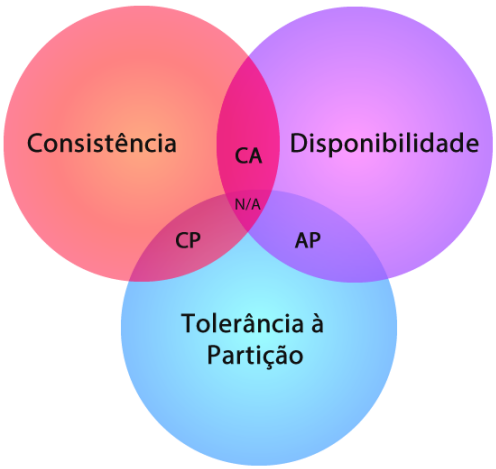


Figura 3.1: Teorema CAP, adaptado de [21].

O teorema, como ilustrado na Figura 3.1, determina ainda que é impossível abrangir todas as três propriedades simultaneamente em um sistema de bancos de dados distribuído. É, portanto, necessário escolher apenas duas delas, de acordo com a finalidade da aplicação em questão.

Sendo escalabilidade horizontal uma das principais premissas dos sistemas NoSQL, logicamente a Tolerância a Particionamento deverá ser atentida - caso contrário não seria um sistema distribuído. Por conseguinte, qualquer banco de dados distribuído precisa escolher entre favorecer a Consistência ou a Disponibilidade [67, 76].

3.2.2 Gerenciamento Transacional

Em um sistema gerenciador de bancos de dados, registros podem ser inseridos, excluídos e atualizados a todo momento. Quanto maior o número de usuários acessando determinado banco, maior será a concorrência entre as operações realizadas, e portanto se vê necessário efetuar a gerência destas transações.

Para garantir a integridade dos dados, a maioria dos sistemas gerenciadores de bancos de dados relacionais se baseiam em transações de propriedade ACID, que devem ser impostas pelo próprio SGBD [48]. São elas:

- **Atomicidade**: garante que todas as alterações no estado do banco de dados serão efetivadas, ou nada será. Caso um erro ocorra, impossibilitando de completar a transação em sua totalidade, o sistema retorna ao estado em que estava antes do início da transação;
- **Consistência**: o banco deve passar de um estado consistente para outro após cada transação. No modelo relacional, isto implica na permamente manutenção do esquema do banco de dados, garantindo que as restrições impostas aos dados não sejam violadas;
- **Isolamento**: a propriedade anterior garante que o estado do banco esteja consistente antes e após as transações. Durante, porém, isto nem sempre é válido. O isolamento oculta alterações não efetivadas de outras transações em execução, impedindo que elas acessem dados que possam levar o banco de dados a um estado não consistente;
- **Durabilidade**: dados de transações efetivadas não podem simplesmente desaparecer do banco. As mudanças efetivadas na base de dados devem persistir aos próximos estados independente de qualquer falha subsequente.

Estas são características desejáveis em uma transação. Todavia, elas não são estritamente necessárias em todos os casos de uso. Aplicações que não implementam transações ACID ainda podem garantir uma consistência eventual, e de acordo com o Teorema CAP, visto na seção anterior, quando a consistência é renunciada, pode-se obter mais disponibilidade e ter maior capacidade em escalar horizontalmente o banco de dados [67].

Assim, surgiu o modelo transacional BASE [68]. Em contraposição a rigidez do modelo ACID, que prima por consistência em todas as operações, o BASE é mais flexível e otimista, aceitando que a consistência da base de dados seguirá o devido curso, e deste modo assegurando a alta escalabilidade que não pode ser obtida através do ACID.

Tabela 3.1: Comparação entre os Modelos Transacionais ACID e BASE [20].

ACID	**BASE**
Forte consistência	Consistência eventual
Baixo particionamento	Alto particionamento
Pessimista	Otimista
Difícil evolução	Fácil evolução

A Tabela 3.1 apresenta as principais diferenças entre os modelos. Como pode ser observado, o modelo ACID fornece baixo particionamento e é considerado pessimista por preocupar-se mais com a segurança e a consistência dos dados do que as aplicações necessitam. Já o modelo BASE provê alta capacidade de particionamento e é considerado otimista pois ao invés de exigir consistência após cada transação, é suficiente que o banco de dados seja eventualmente consistente.

O acrônimo BASE tem como origem os três princípios fundamentais relativos a este modelo [68]:

- **Basicamente disponível** (*Basically available*): a disponibilidade dos dados é provida de acordo com a proposta do Teorema CAP, e atingida pelo suporte parcial às falhas sem que o sistema de modo geral seja comprometido;
- **Estado leve** (*Soft state*): o sistema possui estado volátil, e o valor dos dados armazenados pode mudar pois a consistência não é totalmente garantida;
- **Eventualmente consistente** (*Eventual consistency*): o sistema garante que se não houverem novas alterações, eventualmente a consistência dos dados será retomada, quando a janela de inconsistência for encerrada.

3.3 Tipos e Categorias

É possível encontrar, atualmente, um grande número de bancos de dados NoSQL, divididos em categorias de acordo com a estrutura de dados utilizada. A Figura 3.2 representa a classificação de algumas destas categorias. As modelagens orientadas a Chave-valor, Colunas e Documentos são as três mais populares [25], todas estas utilizando do preceito de agregação de dados em sua estrutura.

Uma outra categoria de bancos não relacionais tem se destacado recentemente devido à alta conectividade dos dados: os baseados em grafos.

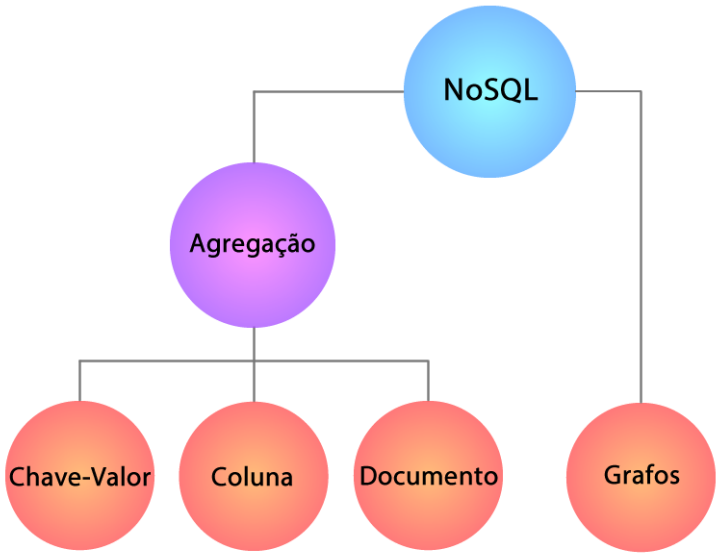


Figura 3.2: Classificação de Sistemas NoSQL.

3.3.1 Banco de Dados Orientado a Chave-Valor

Considerado como o mais simples, este modelo estrutura o banco de dados como uma grande tabela *hash*. Os dados são compostos por uma chave única e pelos valores, suas informações, em forma de um tipo primitivo seja ele inteiro, *string*, vetor ou objeto. Isto elimina a necessidade de um modelo de dados fixo [54].

Nesta abordagem o banco de dados é como um dicionário, onde há uma lista de palavras — chave — e cada palavra possui uma ou mais definições — valor. Assim como no dicionário, as chaves encontram-se ordenadas e, portanto, não é necessário pesquisar todo o banco de dados para encontrar o que se deseja; basta ir direto à chave, tornando a consulta rápida e eficiente [59].

A Figura 3.3 exemplifica a representação de um objeto armazenado em um banco de dados orientado a chave-valor, onde a chave não possui um tipo fixo e pode estar associada a um ou mais valores.

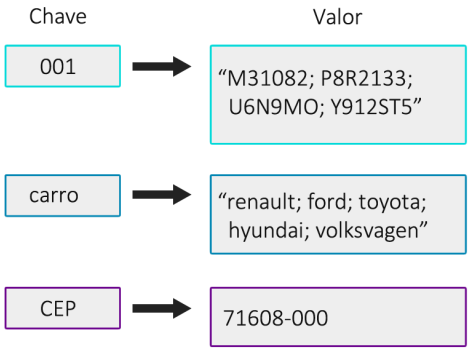


Figura 3.3: Exemplo de Objetos em um Banco de Dados Orientado a Chave-valor.

As operações de leitura nessa base de dados só podem ser executadas através das chaves, não permitindo consultas mais complexas, e são limitadas ao resultado exato. Isto aumenta consideravelmente a velocidade tanto de inserção quanto de recuperação de dados da base [61]. No entanto, há uma alta redundância de informações devido à replicação de dados pelos nós nos sistemas distribuídos que visam garantir disponibilidade.

A principal precursora deste modelo foi a Amazon[1] que em 2007 publicou um artigo sobre o DynamoDB [34], uma solução NoSQL para o gerenciamento de seu sistema global de compras *on-line*. A empresa avaliou que o modelo relacional levaria a uma performance ineficiente e escalabilidade limitada, e optou pelo modelo chave-valor pela alta disponibilidade e devido à maior parte de seus serviços necessitarem apenas de uma chave primária. Além da relação custo/benefício ser mais favorável, o DynamoDB fornece maior desempenho em relação ao modelo relacional, e permite o contínuo crescimento do sistema.

Vários tipos de soluções *open source* para sistemas de bancos de dados orientados a chave-valor surgiram nos últimos anos, mas a maioria delas como o Riak[2] e Voldemort[3] foram inspirados pelo DynamoDB da Amazon e seguem seus preceitos de particionamento, replicação e versionamento dos dados [79].

[1] https://www.amazon.com/
[2] http://basho.com/products/
[3] http://www.project-voldemort.com/

3.3.2 Banco de Dados Orientado a Documentos

Considerado o mais flexível e popular dentre os bancos de dados NoSQL [81], como o próprio nome sugere, o conceito básico deste modelo é uma unidade semi-estruturada de informação: o documento. Este pode ser um arquivo XML, JSON, OpenOffice, ou qualquer outro formato de documento, visto que a base de dados não depende de um esquema rígido definido. Isto possibilita alterações e atualizações na estrutura do documento como a adição de novos atributos, por exemplo, sem comprometer o banco de dados [76].

Toda informação contida no documento é indexada na base de dados, tornando todo o documento pesquisável. Isso significa que, ao contrário dos bancos de dados orientados a chave-valor, nos quais as consultas eram realizadas pela chave, nos bancos de dados em documento, caso determinada propriedade seja conhecida, é possível encontrar todos os documentos que possuam tal propriedade [59].

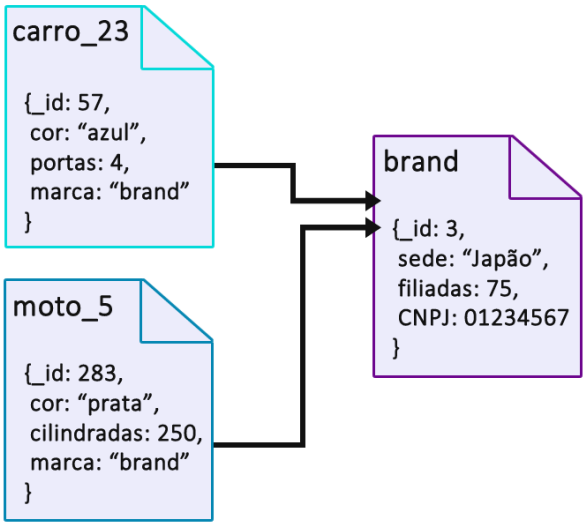


Figura 3.4: Exemplo de Objetos em um Banco de Dados Orientado a Documentos, adaptado de [27].

A Figura 3.4 apresenta um exemplo de objeto em uma base de dados em documentos. Um documento é o correspondente ao registro no modelo relacional, podendo haver uma coleção deles, o que seria equivalente às tabelas, mas sem a rigidez destas [80]. A noção de chave estrangeira também está presente neste modelo, no qual um documento pode ser referenciado como valor de uma chave de outro documento. No modelo relacional, porém, os registros de uma tabela possuem os mesmos campos de atributos, enquanto no banco de dados orientado a documento, cada documento pode ter atributos diferentes de acordo com a necessidade da aplicação [62].

Os principais bancos de dados nesta categoria são o MongoDB[4], desenvolvido em C++, e o Couchbase[5] escrito em C e Erlang.

[4] https://www.mongodb.org/
[5] http://www.couchbase.com/

3.3.3 Banco de Dados Orientado a Colunas

Os bancos de dados orientados a colunas tem o Google como principal pioneiro, quando este decidiu desenvolver sua própria resposta para lidar com o armazenamento e manipulação de sua grande e crescente base de dados. Em 2006 a empresa publicou o artigo sobre o *BigTable* [26] que mais tarde se tornou inspiração para soluções *open source* como Cassandra[6] e HBase[7]. No artigo, os autores acreditavam que o modelo chave-valor proveniente de tabela *hash* distribuída era muito limitado, e que apesar de úteis, os pares chave-valor não deveriam ser a única estrutura disponível aos desenvolvedores. O *BigTable* propôs então um modelo de dados mais complexo e com suporte a dados semi-estruturados.

Algumas das principais características deste modelo de armazenamento e gerenciamento de dados são o particionamento, a indexação e a alta compressão dos dados. A principal desvantagem, no entanto, tem relação às operações de escrita no banco, onde as tuplas devem ser quebradas em seus atributos componentes para que cada um deles seja inserido na base separadamente [12], elevando o tempo das transações.

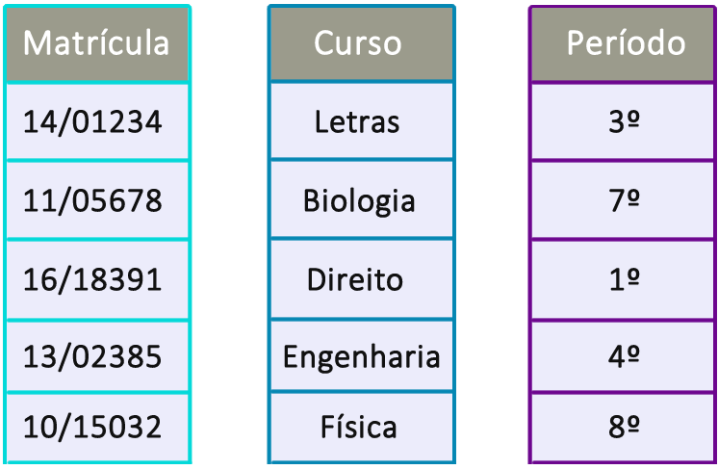


Figura 3.5: Exemplo de Objetos em um Banco de Dados Orientado a Colunas.

A Figura 3.5 apresenta um exemplo de registros armazenados em um banco de dados orientado a coluna, onde observa-se esse problema em que um mesmo registro tem atributos armazenados em colunas separadas.

Este fator torna os bancos de dados orientados a colunas não favoráveis para sistemas do tipo OLTP (*On-Line Transaction Processing* ou Processamento de Transações em Tempo Real) em que a escrita de dados se tornaria um gargalo, mas eficientes em sistemas OLAP (*On-Line Analytical Processing* ou Processamento Analítico em Tempo Real) em que prevalecem as operações de leitura e, portanto, não afetando seu desempenho [66].

[6] http://cassandra.apache.org/
[7] http://hbase.apache.org/

3.3.4 Banco de Dados Orientado a Grafos

Grafos foram descritos formalmente pela primeira vez em 1736 pelo matemático suíço Leonhard Euler, quando este apresentou a solução para o problema conhecido como *as sete pontes de Königsberg*, e que daria origem a teoria dos grafos [83].

O problema descreve a cidade de Königsberg do antigo império Prússio construida tendo o rio Pregel à sua volta. O rio atravessava a cidade de tal forma que a dividia em quatro partes que se interligavam através de sete pontes. A esquematização do problema pode ser observada na Figura 3.6, onde cada parte da cidade A, B, C e D é conectada por pontes identificadas como a, b, c, d, e, f e g.

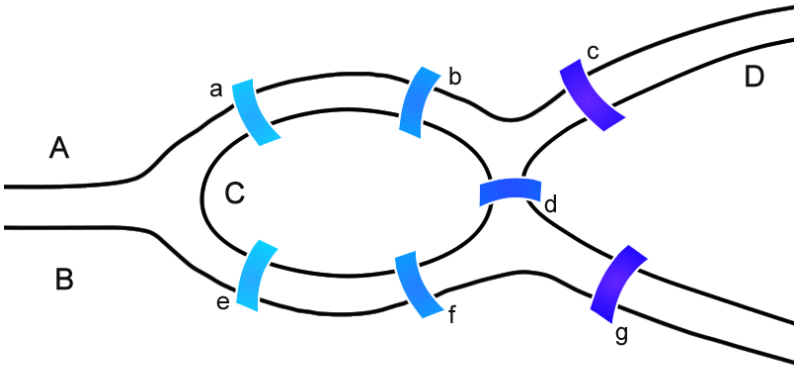


Figura 3.6: Diagrama Ilustrando o Problema das Sete Pontes de Königsberg [83].

O ponto fundamental do desafio era puramente uma questão de trajetória: saber como fazer um *tour* da cidade passando pelas quatro partes e atravessando cada uma das sete pontes, sem andar por uma delas mais de uma vez.

Euler seguiu uma direção diferente das tomadas até então e decidiu abstrair o problema, reduzindo-o a pontos (representando as quatro partes da cidade) e linhas (representando as sete pontes), como mostrado na Figura 3.7, o que viria a ser considerado como o primeiro grafo do mundo [81].

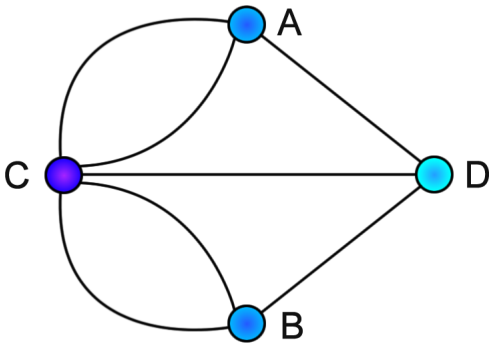


Figura 3.7: Abstração Realizada por Euler no Problema de Königsberg [83].

Sua grande contribuição não foi a resolução em si do problema, mas os métodos utilizados para chegar nela. O problema foi resolvido após Euler aplicar algoritmos matemáticos à sua abstração, provando enfim que tal percurso pela cidade de Königsberg era impossível de ser realizado.

Um grafo é, portanto, como descrito por Van Bruggen, uma representação abstrata de duas ou mais entidades que se relacionam de alguma forma [81]. Matematicamente, um grafo pode ser definido por uma tripla ordenada contendo um conjunto não vazio de vértices, um conjunto não vazio de arestas, e uma função que relaciona cada aresta a um par não ordenado de vértices, sendo representado por

$$G = (V(G), A(G), \psi_{\mathrm{G}})$$

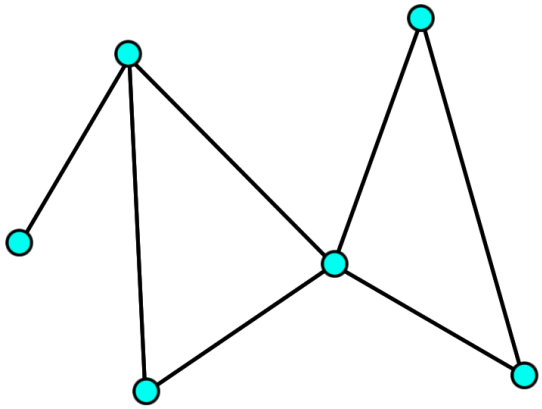
onde $V(G)$ é o conjunto de vértices, $A(G)$ é o conjunto de arestas (ou ligações), e ψ_{G} a função de incidência [19, 78].

Na prática, apenas vértices e arestas não são suficientes para traduzir grafos do mundo real. Muitas vezes deseja-se obter nomes associados a vértices ou um peso e direção vinculados a uma aresta, por exemplo.

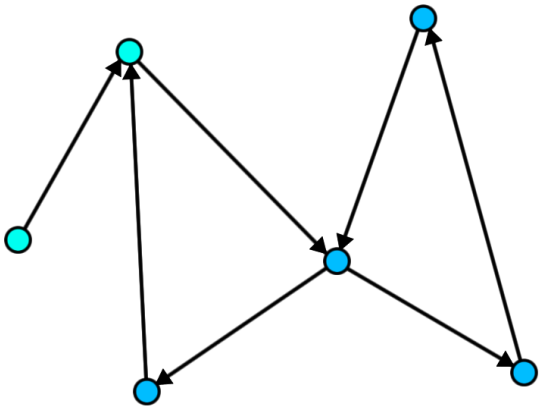
Grafos mais expressivos podem ser criados a partir da adição de características e propriedades aos pontos e linhas originais [72]. A Figura 3.8 mostra alguns dos diferentes tipos de grafos, que são brevemente descritos como:

- **Grafo simples**: um grafo sem arestas paralelas onde uma aresta conecta dois vértices distintos, não permitindo laços (Figura 3.8a);
- **Grafo dirigido**: definido como um par ordenado de vértices, denotando assim a orientação da aresta (Figura 3.8b);
- **Grafo valorado**: possui funções relacionando o conjunto de vértices ou o conjunto de arestas, lhes atribuindo peso (Figura 3.8c);
- **Multigrafo**: permite arestas paralelas, ou seja, dois nós podem estar conectados por mais de uma aresta (Figura 3.8d);
- **Hipergrafo**: generaliza um grafo, com suas arestas ligando uma quantidade arbitrária de vértices (Figura 3.8e);
- **Pseudografo**: denota uma relação reflexiva (Figura 3.8f);

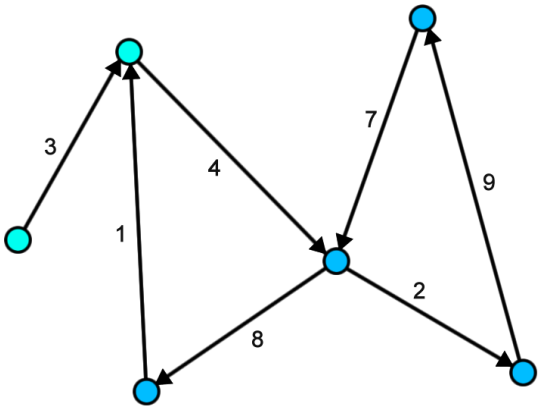
É importante ressaltar que a utilização desses e outros tipos de grafos não é mutuamente exclusiva. A combinação deles é fundamental para que se consiga representar com expressividade as características do domínio de forma apropriada.



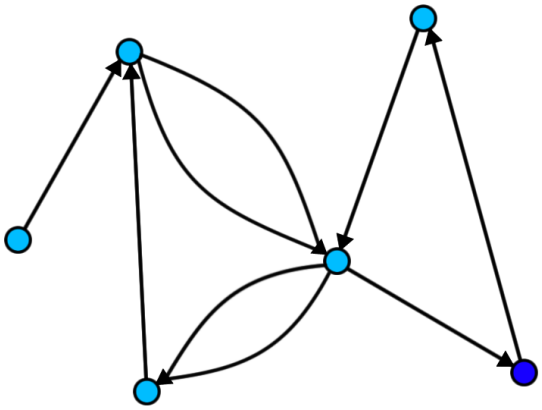
(a) Grafo simples



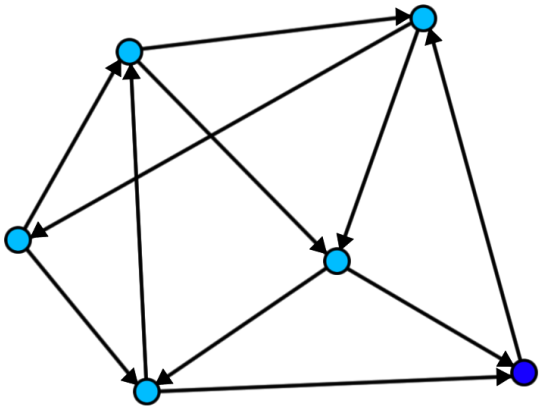
(b) Grafo dirigido



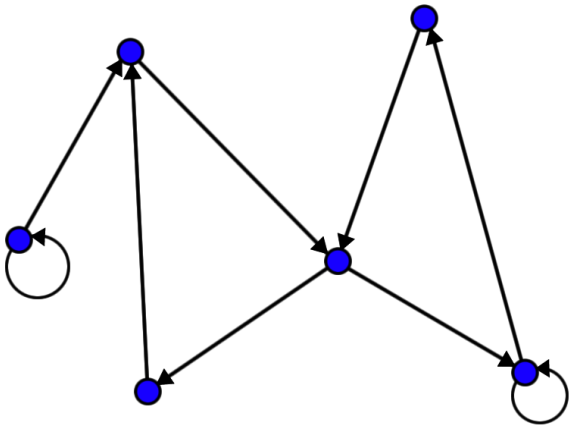
(c) Grafo valorado



(d) Multigrafo



(e) Hipergrafo



(f) Pseudografo

Figura 3.8: Representação de Alguns Tipos de Grafos, adaptado de [72].

Assim, a teoria dos grafos serve como base para os bancos de dados orientados a grafos, que utilizam de suas premissas para descrever e armazenar dados altamente conectados. Os vértices – aqui chamados de nós – simbolizam os registros; e as arestas expressam o relacionamento entre os nós. Além disso, é utilizado principalmente o tipo de grafo de propriedades ou atributos [60], permitindo que ambos elementos possam conter especificações de suas características. A Figura 3.9 representa este tipo de grafo, em que os nós possuem atributos próprios, e as arestas que ligam os nós possuem atributos descrevendo a relação entre eles.

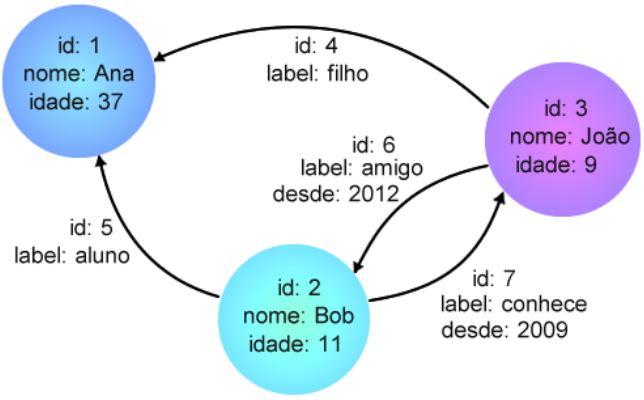


Figura 3.9: Exemplo Simples de Grafo de Propriedades.

Embora possam parecer uma tecnologia atual, os bancos de dados orientados a grafos despontaram na década de 80 simultaneamente com os modelos orientados a objetos, mas apenas recentemente, com o advento das redes sociais juntamente com a necessidade de gerenciar os dados que estas provêm, assim como novas aplicações em biologia e química, fizeram com que os grafos reconquistassem destaque [15].

Uma das principais vantagens desse modelo é que, ao contrário da maioria dos NoSQL vista anteriormente, os bancos de dados orientados a grafos têm suporte completo às transações ACID de forma nativa [25], tornando operações de inserção e de manipulação de dados confiáveis e consistentes, garantindo maior segurança e credibilidade à aplicação [71].

Outra característica relevante é o fato dos relacionamentos serem armazenados explicitamente na base de dados, e não inferidos por uma chave externa ou determinados por operações de *join*, conferindo-lhes o mesmo nível de expressividade dado aos nós [81].

A manipulação dos dados é realizada por meio de operações inerentes à estrutura de grafos como percurso, vizinhança, padrões, conectividade, e funções de estatísticas como diâmentro, profundidade, centralidade, etc [45].

Segundo Robinson et al. [71], os bancos de dados orientados a grafos são mais indicados quando as relações entre os nós são mais importantes do que os dados neles contidos. Em uma rede social, por exemplo, na geração de recomendação de amigos; ou recomendação de produtos em um site de loja; e até mesmo na condução de investigações criminais. Todos esses casos têm enfoque no percurso levado de um nó a outro, no trajeto percorrido até determinado ponto.

Os bancos de dados em grafo são otimizados para situações em que a detecção e o reconhecimento de padrões sejam essenciais. Para consultas amplas ou de baixa comple-

xidade, ou análise dos valores de um conjunto de registros, talvez um outro tipo de banco de dados seja mais recomendado [61].

Ao contrário dos outros tipos de NoSQL, os bancos de dados em grafo apresentam limitações de escalabilidade em múltiplos servidores devido a alta conectividade entre os nós. Os dados podem ser replicados entre servidores, com o objetivo de aumentar o desempenho em operações de leitura e consulta, mas operações de escrita e consultas envolvendo múltiplos nós podem ser complexas de implementar [59].

Os principais sistemas gerenciadores orientados a grafos são o Neo4j[8], Titan[9] e o OrientDB[10], que apesar de terem sido implementados sob abordagens diferentes, compartilham de conceitos básicos semelhantes, como a utilização do modelo de grafo de propriedades.

Titan

O Titan é um banco de dados orientado a grafo distribuído e otimidado para o armazenamento e consultas de grafos representados em agrupamentos de máquinas. Ele foi projetado para suportar o processamento de grafos tão grandes que necessitassem de poder computacional além do que uma única máquina poderia fornecer [7]. Algumas de suas principais funcionalidades são:

- código aberto e livre para uso irrestrito, inclusive comercial;
- operações de processamento concorrente;
- suporte a transações ACID;
- escalabilidade linear.

OrientDB

Lançado em 2010, o OrientDB é um projeto *open source* de banco de dados multimodelo, suportando abordagens orientadas a grafo, documento, chave-valor e objeto, mas os relacionamentos são gerenciados como em um banco de dados em grafo, utilizando conexões diretas entre os registros. Ele realiza o armazenamento de dados não estruturados, semi, e estruturados, e permite o uso do SQL como linguagem de consulta, fornecendo ainda uma linguagem própria, para manipulação de árvores e grafos, baseada no SQL a fim de reduzir a curva de aprendizado dos iniciantes em bancos de dados em grafo [11].

Atualmente, são ofertadas duas versões do OrientDB, sendo a edição empresarial uma extensão da edição comunitária [77].

- **Edição Comunitária** (*Community Edition*)
  - aberta e livre para uso irrestrito, inclusive comercial;
  - suporte a transações ACID;
  - replicação.

[8] http://www.neo4j.com/
[9] http://www.titan.thinkaurelius.com/
[10] http://www.orientdb.com/

- **Edição Empresarial** (*Enterprise Edition*)
  - suporte técnico disponível 24 horas;
  - análise métrica dos dados;
  - *backup* e restauração de dados.

Neo4j

Lançado em fevereiro de 2010, o Neo4j, como muitas soluções NoSQL é um projeto *open source* que nasceu da necessidade de resolução de problemas particulares que não eram bem atentidos pelos sistemas relacionais tradicionais, e fizeram com que os desenvolvedores buscassem uma nova abordagem.

Inicialmente, ele não havia sido projetado para ser um sistema gerenciador de banco de dados em grafos como é hoje. Seu foco era criar camadas de abstração em grafos, mas utilizando o *MySQL*[11] e outros SGBD relacionais para tal finalidade [81]. Após um tempo, essa estratégia se provou insuficiente, e a equipe do projeto tomou a decisão de abandonar completamente o modelo relacional para criar uma base de dados que tivesse o grafo como elemento fundamental. Toda a estrutura do sistema foi remodelada e otimizada para trabalhar diretamente com dados em forma de grafo de modo nativo, aumentando assim o seu desempenho.

O Neo4j compartilha de muitas qualidades dos SGBD relacionais atuais, utilizando um modelo de dados completamente diferente que entretanto se mostra adequado a conjuntos de dados altamente conectados, sendo mais rápido do que os sistemas tradicionais em alguns casos [82]. Ele apresenta também conformidade com transações ACID, o que geralmente não é suportado por sistemas NoSQL.

Originalmente foi desenvolvido em Java, porém, oferece suporte a várias linguagens como Python, Ruby e PHP. Ele encontra-se na versão 2.3.1 e possui atualmente duas opções de licenciamento:

- **Edição Comunitária** (*Community Edition*)
  - aberta e livre para uso, inclusive comercial;
  - suporte da comunidade através do Stack Overflow[12], Google Groups[13], etc;
  - aplicação básica completa e de alta performance.
- **Edição Empresarial** (*Enterprise Edition*)
  - equipe de suporte profissional disponível 24 horas;
  - acesso direto aos desenvolvedores;
  - opções avançadas de gerenciamento;
  - processamento distribuido e maior poder de escalabilidade.

[11] https://www.mysql.com/
[12] http://www.stackoverflow.com/
[13] https://www.groups.google.com/forum/

Poder interagir com os grafos de forma visual permite a extração de informação de maneira mais simples, uma vez que é mais fácil entender imagens do que tabelas. Outro fator importante da visualização dos grafos é o fácil reconhecimento de padrões nos dados, auxiliando nas tomadas de decisão.

Em 2013 a equipe desenvolvedora do Neo4j lançou o Neo4j *Browser*, um ambiente híbrido entre ferramenta de consulta e área de desenvolvimento. Ele fornece funcionalidade que se espera em uma exploração interativa de dados em grafos, permitindo salvar consultas e estilizar o visual [81].

A Figura 3.10 representa este ambiente, em que na barra superior são executadas as consultas, e o resultado é apresentado logo abaixo em formato de grafo.

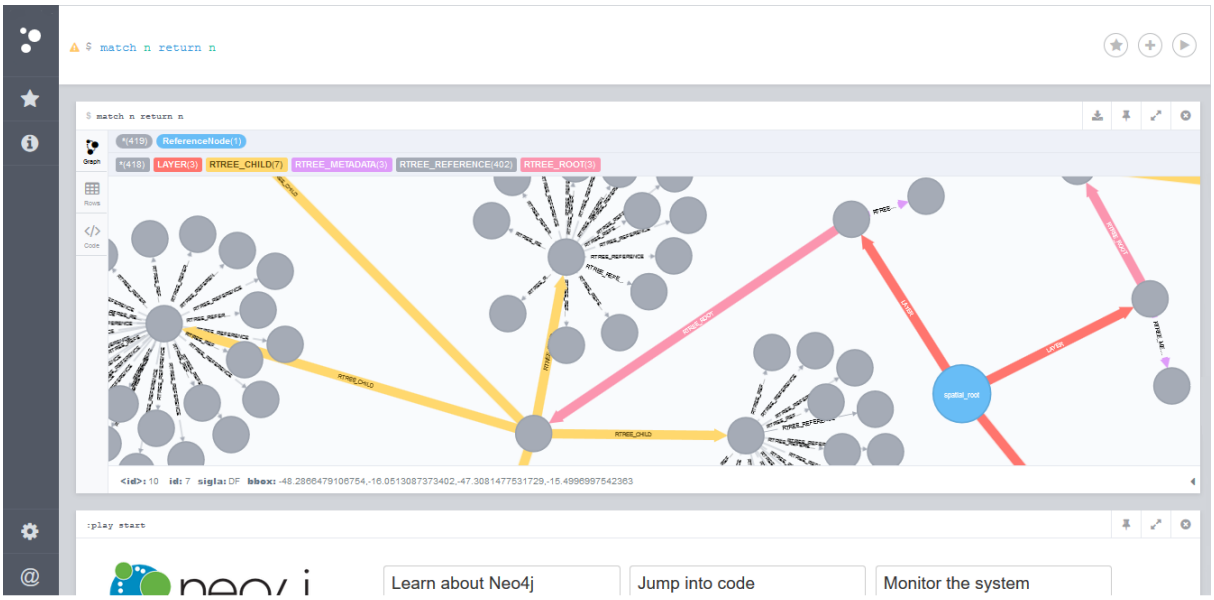


Figura 3.10: Interface Neo4j *Browser*.

No início, utilizar o Neo4j em um projeto era uma tarefa difícil, se comparado ao tradicional banco relacional. A execução das consultas era complexa, acarretando em perda de performance [64]. Para resolver esse problema, o Neo4j introduziu o *Cypher*.

Cypher é uma linguagem de consulta declarativa que permite realizar casamento de padrões em grafos de forma expressiva e eficiente. Desenhada para ser uma linguagem amigável e de fácil entendimento, ela enfatiza o quê retornar do grafo, não em como retornar, em oposição às linguagens de consultas imperativas existentes [81].

A linguagem foi inspirada por diversas abordagens e práticas estabelecidas para se realizar uma consulta efetiva. Expressões como *where* e *order* vieram do SQL, enquanto algumas estruturas semânticas tiveram de influência linguagens como *Python* e *Haskell* [4].

3.4 Popularidade

O site *DB-Engines*[14], uma iniciativa da empresa austríaca de consultoria *solid IT*[15], tem como objetivo coletar e apresentar informações a respeito dos sistemas gerenciadores de bancos de dados, englobando tanto os sistemas SGBD relacionais quanto os NoSQL.

A Figura 3.11 mostra o gráfico de popularidade e adoção dos sistemas SGBD publicado pelo site em 2013, no qual pode-se observar que o modelo relacional dominava o mercado representando quase 91% do total, enquanto os sistemas não relacionais mencionados neste capítulo somavam 7,4%.

Três anos depois, no entanto, como ilustrado na Figura 3.12, os sistemas NoSQL citados aqui conquistaram uma parcela maior do mercado, totalizando agora 13,7%, contra 81,8% do modelo relacional.

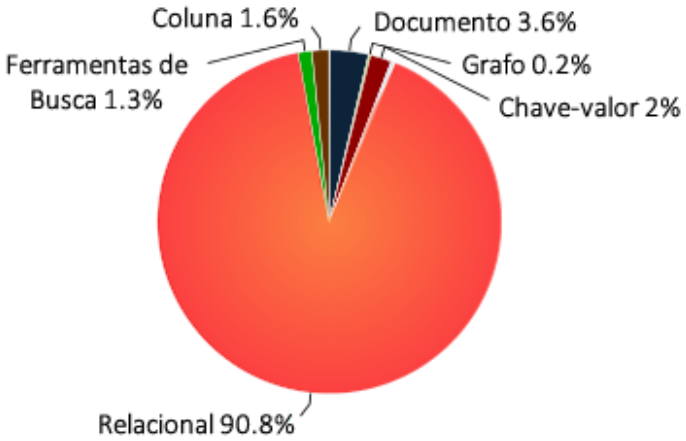


Figura 3.11: Gráfico de Popularidade dos Modelos SGBD Realizado em Novembro de 2013, adaptado de [14].

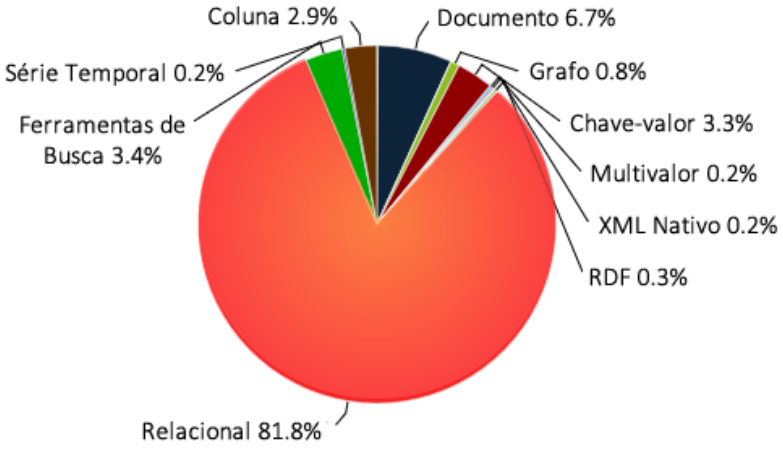


Figura 3.12: Gráfico de Popularidade dos Modelos SGBD Realizado em Junho de 2016, adaptado de [9].

[14] http://www.db-engines.com/
[15] http://www.solid-it.at/

Outro tipo de análise disponibilizada pelo *DB-Engines* é o de variações na popularidade em cada categoria de SGBD. Na Figura 3.13 pode-se observar que apesar dos bancos orientados a grafos apresentarem o maior crescimento no ano de 2013, os demais sistemas NoSQL também mostraram crescimento significativo.

Entretanto, como ilustrado na Figura 3.14, enquanto os demais modelos não relacionais mantiveram sua taxa de crescimento, a partir de 2014 os SGBD baseados em grafos dispontaram, apresentando crescimento cerca de 50% superior em relação aos concorrentes.

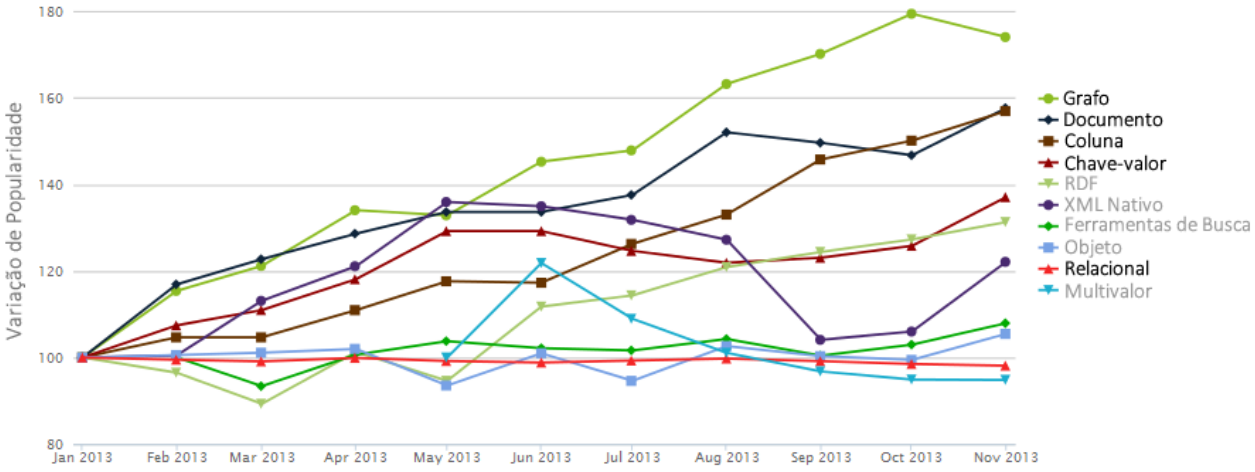


Figura 3.13: Gráfico da Variação de Popularidade dos Modelos SGBD Realizado em Novembro de 2013 [14].

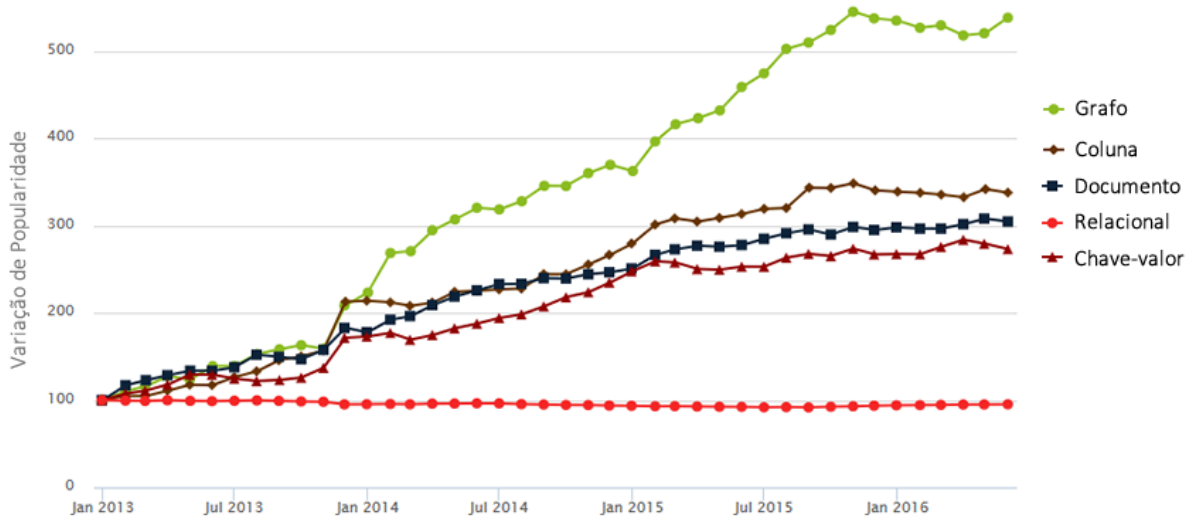


Figura 3.14: Gráfico da Variação de Popularidade dos Modelos SGBD Realizado em Junho de 2016 [9].

3.5 Conclusão

Neste capítulo foram apresentados os bancos de dados NoSQL, seu histórico, suas características e as abordagens que levaram ao surgimento das quatro vertentes pricipais aqui discutidas: chave-valor, colunas, documentos e grafos. Embora alguns modelos existam desde os anos 80, foi nos últimos dez anos que o movimento NoSQL realmente ganhou força, abrangendo particularidades não suportadas pelo modelo relacional, e até mesmo conquistando usuários deste através de sua alta escalabilidade horizontal e capacidade de gerenciar dados em larga escala.

Apesar do modelo relacional ainda mostrar força e dominar o mercado de banco de dados, ele vem perdendo espaço para os sistemas NoSQL a cada ano, cenário que pode se manter devido à adesão ao modelo não relacional pelas grandes empresas. Contudo, o modelo relacional ainda se mostra uma solução mais geral e abrangente que seus concorrentes, garantindo-lhe certa vantagem.

Como visto anteriormente no Capítulo 2, a crescente demanda de dados geográficos, não apenas em decorrência de aplicações científicas mas também comerciais [13], provocou a necessidade de sistemas de bancos de dados capazes também de gerenciar e analisar dados geoespaciais. Funcionalidade já presente nos principais bancos de dados relacionais, os emergentes NoSQL agora se adaptam para fornecer suporte a dados geográficos, mantendo-se como uma alternativa ao modelo relacional, tema a ser abordado no próximo capítulo.

Capítulo 4

Bancos de Dados Geográficos NoSQL

Este capítulo apresenta funcionalidades de alguns bancos de dados NoSQL que oferecem suporte aos dados geográficos, e aborda as características desejáveis em um banco de dados geográficos NoSQL.

4.1 Introdução

Como abordado no capítulo anterior, a quantidade de dados gerada atualmente tornou-se um desafio em questão de escalabilidade aos bancos de dados relacionais, fazendo com que empresas buscassem alternativas ao esquema rígido do modelo relacional para o armazenamento de dados mais complexos e menos estruturados.

A grande ascenção de redes sociais como Facebook[1], Twitter[2], Instagram[3] e Foursquare[4] trouxe consigo uma nova necessidade aos bancos de dados NoSQL: o armazenamento de dados geográficos [33]. Isto porque estas aplicações da *web* tem como característica em comum o georreferenciamento de dados, seja para indicar os locais por onde você passou, aonde aquela foto foi tirada, ou simplesmente para indicar a origem de sua postagem.

Apesar de inicialmente não terem sido diretamente projetados para tratar com dados espaciais, os sistemas NoSQL já utilizados por trás das redes sociais mencionadas, precisaram evoluir para atender essa nova tendência e oferecer suporte aos dados geográficos [74].

4.2 Tipos e Categorias

Batista et al. em [33], apresentam sete características desejáveis em um sistema NoSQL que suporte dados geoespaciais:

- **Conceitos básicos**: é importante saber se o sistema NoSQL permite o uso de diferentes sistemas de referência espacial, de acordo com os requisitos da aplicação;

[1] https://www.facebook.com/
[2] https://www.twitter.com/
[3] https://www.instagram.com/
[4] https://www.foursquare.com/

qual a dimensionalidade dos dados suportados (1D, 2D, 3D); e por fim, qual o modelo de dados utilizado internamente para seu armazenamento (baseado em chave-valor, colunas, documento ou grafo);

- **Indexação**: é importante saber como a indexação dos dados espaciais é realizada, pois determinadas técnicas como árvores-R são reconhecidas por agilizar o tempo de processamento [47];

- **Tipos de dados vetoriais**: é importante saber quais os tipos de dados suportados pelo sistema NoSQL (ponto, linha, polígono, multi-ponto, multi-linha, etc.). O Consórcio Geoespacial Aberto (Open Geospatial Consortium - OGC)[5] propôs um padrão para tipos de dados vetoriais que especifica modelos de acesso e armazenamento para dados geográficos bidimensionais e suas funções espaciais correspondentes;

- **Entrada e Saída de dados**: é importante saber se os principais formatos de entrada e de saída de dados geográficos são suportados pelo sistema NoSQL, como:

  - *GML (Geography Markup Language)*: traduzido como linguagem de marcação de geografia, este formato foi definido pelo Consórcio Geoespacial Aberto e utiliza XML para expressar características geográficas;
  - *WKT (Well-Known Text)*: o formato de texto reconhecido também foi especificado pelo OGC, e representa informação geográfica de forma textual;
  - *WKB (Well-Known Binary)*: similar ao WKT, o formato de binário reconhecido representa informação geográfica utilizando binários;
  - *SVG (Scalable Vector Graphics)*: traduzido como gráficos vetoriais escaláveis, é o formato proposto pela W3C[6] baseado na linguagem XML para descrever de forma vetorial imagens e gráficos bidimensionais;
  - *JSON (JavaScript Object Notation)*: traduzido como notação de objetos JavaScript, possui a variação GeoJSON projetada para a representação de fenômenos geográficos juntamente com seus atributos não espaciais. O formato GeoJSON difere dos outros padrões para dados geográficos por ser um padrão aberto e não ser mantido por uma organização formal;
  - *SHP (Shapefile)*: é o formato proprietário de dado geoespacial da empresa ESRI[7], e considerado o formato padrão;

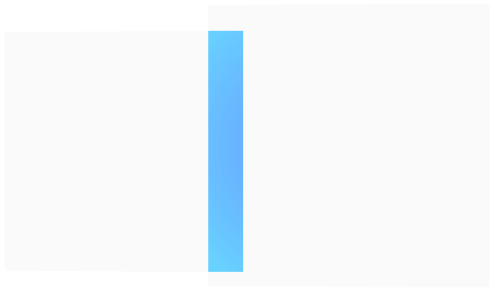
- **Funções escalares**: é importante saber se o sistema NoSQL dispõe de funções de análise métrica como:

  - *length()*: retorna o comprimento da geometria;
  - *area()*: retorna a área da geometria;
  - *distance(theGeom)*: retorna a distância entre as geometrias;

[5] http://www.opengeospatial.org/
[6] http://www.w3.org/
[7] http://www.esri.com/

(a) União

(b) Diferença



(c) Interseção

Figura 4.1: Representação de Funções Geoespaciais Aplicadas a Conjuntos de Dados.

- **Funções de conjunto**: funções de análise executadas em um conjunto de dados espaciais como:
  - *union(theGeom)*: retorna a geometria que representa a união das geometrias (Figura 4.1a);
  - *symDifference(theGeom)*: retorna a geometria representando as partes das geometrias que não intersetam (Figura 4.1b);
  - *intersection(theGeom)*: retorna a geometria que representa a interseção das geometrias (Figura 4.1c);

(a) Identidade

(b) Fronteira

(c) Disjunção

(d) Contém

Figura 4.2: Representação de Funções Geoespaciais de Análise Topológica.

- **Funções espaciais**: é importante saber se o sistema NoSQL provê funções básicas de análise topológica, tais como:
  - *equals(theGeom)*: dada uma geometria, indica se *theGeom* representa a mesma geometria (Figura 4.2a);
  - *touches(theGeom)*: indica se as fronteiras das geometrias intersetam, mas não seus interiores (Figura 4.2b);
  - *disjoint(theGeom)*: indica se as geometrias não compartilham de espaço em comum, isto é, elas não se intersetam (Figura 4.2c);
  - *within(theGeom)*: indica se a geometria *theGeom*) está totalmente contida dentro da outra (Figura 4.2d);

Dos tipos de bancos de dados NoSQL apresentados no Capítulo 3, é possível encontrar, pelo menos, um *software* de cada categoria que forneça algum tipo de suporte a dados geográficos, seja de forma nativa ou por meio de uma extensão. Dentre eles destacam-se o Couchbase, MongoBD e Neo4j:

- **Couchbase**: recomendado para casos de análise multidimensional, dados geográficos, e a combinação de ambos. Suporta o formato GeoJSON, permitindo a representação de geometrias através de pontos, linhas e polígonos. As consultas espaciais são realizadas via REST e podem incluir *bounding boxes* para sua delimitação [6];

- **MongoDB**: o suporte a dados geográficos é dado através do MongoDB Geospatial Indexes[8], que provê dois índices para a consulta e manipulação de dados espaciais: 2D, que utiliza geometria plana, e o 2D*Sphere*, que utiliza geometria esférica no retorno dos resultados [5]. Inicialmente permitia apenas a representação de pontos, porém atualizou sua estrutura para oferecer suporte ao formato GeoJSON, permitindo também a representação de demais tipos de geometria como linhas e polígonos. Possibilita ainda a execução de consultas espaciais envolvendo inclusão, interseção e proximidade. Uma desvantagem é não oferecer ferramentas para importação dos principais arquivos de dados geográficos, e os dados devem ser convertidos em GeoJSON para a inserção;

- **Titan**: suporta apenas geometrias do tipo ponto para indexação, e a busca por pontos dentro de uma geometria é única função espacial oferecida [7];

- **OrientDB**: o suporte a dados geográficos é obtido pelo *plug-in* externo Lucene Spatial[9], e apesar de não possuir uma ferramenta para importação de dados, o OrientDB permite a inserção de geometrias como ponto, linha, polígono, multi-ponto, multi-linha e multi-polígono através de SQL ou Java. A indexação, porém, é limitada a geometrias do tipo ponto. Oferece funções espaciais como identidade, distância, disjunção e interseção [10].

- **Neo4j**: o armazenamento e manipulação de dados geográficos no Neo4j é obtido através do Neo4j-Spatial[10], um *plug-in* espacial desenvolvido em Java. Ele oferece ferramentas automatizadas de importação para arquivos *shapefile* e OpenStreetMap, e possibilita a exportação dos dados no formato *shapefile*. Ele realiza a indexação dos dados utilizando uma árvore-R para otimização das consultas espaciais, que podem envolver funções topológicas como interseção, disjunção, sobreposição, distância, fronteira e proximidade. Além de permitir que os dados geográficos sejam indexados durante a inserção no banco, com o Neo4j-Spatial é possível adicionar propriedades espaciais a dados já armazenados. Suporta os principais tipos de geometria como ponto, linha, polígono, multi-ponto, multi-linha e multi-polígono [1].

Apesar de desejáveis, algumas das características apresentadas ainda não estão implementadas na maioria dos NoSQL abordados, entretanto, isto tende a mudar devido ao crescente interesse pela manipulação e armazenamento de dados geográficos em sistemas NoSQL. Algumas pesquisas foram feitas nos últimos anos para apurar o suporte destas ferramentas a este tipo de dado.

McCarthy em [58] realiza comparações entre um banco de dados relacional e o MongoDB utilizado para verificar se o modelo NoSQL é uma alternativa em relação à dados geográficos. Os resultados mostraram, no entanto, que apesar de apresentar bom desempenho em determinadas consultas, o MongoDB ainda não estava preparado o suficiente para as funções espaciais.

[8] https://www.docs.mongodb.org/manual/applications/geospatial-indexes/
[9] https://lucene.apache.org/core/4_0_0/spatial/
[10] http://neo4j-contrib.github.io/spatial/

Queiroz et al. em [32] apresentam uma revisão crítica da literatura de bancos de dados geográficos, a fim de verificar se os modelos NoSQL podem suprir as lacunas entre os sistemas relacionais e as necessidades das novas aplicações comerciais. Os autores concluem que os NoSQL estão introduzindo suporte ao armazenamento e recuperação de dados espaciais, e apesar de ainda não apresentarem concorrência às ferramentas existentes do relacional, apresentam crescente evolução na área de dados geográficos.

Batista et al. em [33] avaliam quatro bancos de dados NoSQL — CouchDB, MongoDB, BigTable e Neo4j — em relação ao suporte de dados geográficos. Foi observado que estes sistemas precisam de melhorias significativas no que diz respeito aos tipos de dados geográficos e funções espaciais, porém, os SIG podem se beneficiar de suas principais características como alta escalabilitade e armazenamento de dados não estruturados. Após as comparações, o Neo4j foi eleito como a melhor alternativa NoSQL em grafo para o armazenamento e a manipulação de dados geográficos e, portanto, foi selecionado para a realização das análises comparativas, apresentadas no próximo capítulo.

Capítulo 5

Análise de Dados Geográficos em Banco de Dados Relacional e Baseado em Grafo

Neste capítulo é apresentado um estudo comparativo com o objetivo de avaliar o suporte e o desempenho de um banco de dados geográfico orientado a grafo em relação à um banco de dados baseado no modelo relacional. Dentre os critérios utilizados estão a inserção, o armazenamento e a visualização de dados, o desempenho das consultas espaciais, e o suporte e usabilidade do sistema.

5.1 Implementação

Como visto no capítulo anterior, existem algumas características desejáveis em um sistema NoSQL com suporte a dados geográficos, e o Neo4j-Spatial é o que melhor as atende. Um estudo de caso foi implementado para avaliar estas características em relação a uma abordagem relacional.

5.1.1 Dados

Para este projeto foram selecionados dados fornecidos pelo Governo Federal de forma gratuita no formato *shapefile*. O formato, além de ser o mais comum na distribuição de dados espaciais [63] e amplamente utilizado, permite a representação de dados vetoriais para armazenar a posição, o formato, e as propriedades de feições geográficas através de geometrias simples como pontos, linhas, polígonos e suas coleções — multi-ponto, multi-linha e multi-polígono [37]. Para verificar o suporte das aplicações, um *shapefile* de cada tipo de geometria citado foi utilizado.

O Instituto Brasileiro de Geografia e Estatística - IBGE disponibiliza um portal de mapas[1], desenvolvido para facilitar o acesso e a visualização dos mais de 20 mil mapas produzidos pelo Instituto. No portal é possível encontrar seções para mapas escolares, político-administrativos, físicos, temáticos e interativos, onde foi obtido o arquivo no formato *shapefile* contendo 247 registros com os municípios de Goiás e Distrito Federal repre-

[1] http://www.mapas.ibge.gov.br/

sentados como pontos, e o arquivo dos estados brasileiros com 27 registros representados como polígonos.

O Departamento Nacional de Infraestrutura de Transportes - DNIT conta com o setor DNITGeo[2] que tem como objetivos elaborar, estruturar e manter sua base de dados geográficos sobre infraestrutura de transportes do Brasil. No *site* do DNITGeo foi selecionado o arquivo *shapefile* disponibilizado com a malha rodoviária do Distrito Federal contendo 128 registros em que as rodovias estão representadas como linhas.

5.1.2 Funcionalidades

O Neo4j-Spatial é um *plug-in* que permite a realização de operações espaciais em dados armazenados em grafo dentro do Neo4j. Ele pode ser utilizado através de uma API em Java, ou diretamente via REST. Dentre as principais funcionalidades estão ferramentas para importação de arquivos *shapefile* e *OpenStreetMap*, suporte aos principais tipos de geometrias definidos pelo OGC, suporte a funções topológicas, e a indexação dos dados utilizando uma árvore-R para otimização das consultas espaciais.

Apesar da API em Java fornecer suporte a uma maior variedade de funções topológicas e operações espaciais mais complexas, ela requer um conhecimento prévio da linguagem, e sua única documentação são as classes de teste contidas no repositório do GitHub[3]. A interface REST provê suporte às operações básicas como criar uma camada espacial, criar, atualizar e adicionar geometrias à camada, procurar por geometrias dentro de uma distância, dentro de um polígono, e pelas geometrias mais próximas. Ainda que mais simples, esta abordagem traz maior aproveitamento da infraestrutura *web* já existente e reduz o esforço de aprendizado, resultando na consequente maior facilidade de desenvolvimento, uma vez que as chamadas REST independem da linguagem de programação utilizada [65].

O PostGIS[4] é uma extensão do SGBD objeto-relacional PostgreSQL[5], que permite que informações geográficas sejam armazenadas no banco de dados e trabalhadas através de funções para análise e processamento de dados espaciais. O PostGIS também realiza indexação dos dados utilizando uma árvore-R e suporta todos os objetos e funções espaciais definidos no *Simple Features Specification for SQL*, uma especificação que determina padrões para tipos de objetos geográficos e as funções necessárias para manipulá-los utilizando SQL com objetivo de garantir a consistência de seus meta-dados. O PostGIS estende este padrão oferecendo suporte a coordenadas em três e quatro dimensões [70].

Considerações

Embora seja o banco de dados NoSQL com maior suporte a dados geográficos, o Neo4j-Spatial ainda não possui um conjunto de funcionalidades espaciais tão completo quanto o PostGIS, e o fato destas funcionalidades só poderem ser acessadas externamente à interface Neo4j *Browser* através da linguagem Java ou REST apresenta-se como uma desvantagem em relação ao *software* relacional, que permite acesso às suas funcionalidades, geográficas ou não, através de uma mesma interface.

[2] http://www.dnit.gov.br/planejamento-e-pesquisa/dnit-geo
[3] https://www.github.com/neo4j-contrib/spatial
[4] http://www.postgis.net/
[5] http://www.postgresql.org/

5.1.3 Inserção de Dados Geográficos

A versão da API em Java do Neo4j-Spatial inclui uma ferramenta para importação de arquivos *shapefile* de forma automatizada, na qual uma nova camada é criada para cada arquivo e as geometrias são armazenadas no formato WKT. Após realizar alguns testes de importação com diferentes arquivos *shapefile*, foi observado que a função que transforma a geometria para WKT não está implementada de forma correta, e os pares de latitude e longitude são armazenados sem distinção alguma, não sendo possível determinar as geometrias e, portanto, fazendo com que a realização de consultas espaciais seja inviabilizada.

Como abordagem alternativa, foram gerados arquivos CSV com o auxílio do PostgreSQL para cada um dos *shapefile* selecionados, e um *script* em Python foi utilizado para a importação dos dados a partir do arquivo gerado.

```
1   url = "http://localhost:7474/db/data/ext/SpatialPlugin/graphdb/
        addEditableLayer"
    params = {"layer": "estados", "format": "WKT",
       "nodePropertyName" : "wkt"}
4   r = requests.post(url, data=json.dumps(params), headers=headers)

    url = "http://localhost:7474/db/data/node"
7   params = {"gid": row[0], "sigla": row[1], "nome": row[2], "wkt": row[3]}
    r = requests.post(url, data=json.dumps(params), headers=headers)
    node = r.json()['self']
10
    url = "http://localhost:7474/db/data/ext/SpatialPlugin/graphdb/
        addNodeToLayer"
    params = {"layer": "estados", "node": node}
13  r = requests.post(url, data=json.dumps(params), headers=headers)
```

Script 5.1: Script Python para Importação de Dados no Banco a Partir do Arquivo CSV.

O *script* 5.1 cria na linha 4 uma nova camada *estados* que será indexada utilizando uma árvore-R. Além disso, gera os nós na linha 9 armazenando a respectiva geometria no formato WKT enquanto os demais atributos são adicionados como propriedades de cada nó. Por fim, na linha 13 os nós são adicionados à camada inicialmente criada, que poderá ser consultada por meio das operações espaciais suportadas. *Scripts* similares foram utilizados para a importação dos demais arquivos.

Figura 5.1: Fluxo Seguido para a Inserção dos Dados no Neo4j.

Os passos executados para a inserção dos dados no Neo4j estão representados na Figura 5.1, na qual inicialmente foram selecionados os arquivos *shapefile* e a partir deles foram gerados arquivos CSV através do PostgreSQL. Por fim, um *script* em Python foi utilizado para inserir no Neo4j os dados contidos nestes arquivos.

A Figura 5.2 ilustra como o nó fica estruturado após a inserção, no banco de dados, por meio do *script* Python, onde é possível distinguir claramente os pares de latitude e longitude na definição da geometria.


```
  "data" : {
    "sigla" : "CO",
    "bbox" : [ -61.6321641413005, -24.0695098951066, -45.9068610096409, -7.34865991285703 ],
    "wkt" : "MULTIPOLYGON (((-47.3086092747636 -16.0354909115716, -47.310945713715 -16.0408563658703, -47.3081477531729 -16.04
7.3516262317842 -16.1315069800767, -47.3458677638365 -16.1400558897167, -47.349928726151 -16.147674864342, -47.3441962375778 -
, -47.3288122579025 -16.2159414689423, -47.322015777211 -16.2308633817431, -47.3284926980343 -16.2486552709708, -47.3304751588
5019151, -47.4244330758395 -16.4228030000436, -47.428652516737 -16.4330394713577, -47.4313934732827 -16.4396894180425, -47.429
513367815098, -47.4130684010395 -16.5476667876498, -47.4054334496398 -16.5552352559324, -47.4119229159376 -16.5580372263086, -
3 -16.5927239444595, -47.3425063677534 -16.5977003807963, -47.3365149197183 -16.5975173823102, -47.338033400536 -16.6033798728
61673 -16.6463950307108, -47.2778692998932 -16.6477534928743, -47.2776833038049 -16.6497879895019, -47.2767892478985 -16.65959
89865213 -16.7339189682295, -47.2260929902808 -16.7412299232606, -47.2340619431037 -16.7442264128025, -47.2355434081441 -16.75
201223427125 -16.874992122716, -47.200460418378 -16.8749156365171, -47.1579055877418 -16.9642385495756, -47.1512051297746 -16.
, -47.1572179493527 -17.033501213547, (...), -60.3626870425305 -13.2961288228903)))",
    "gid" : "5",
    "nome" : "Centro-Oeste",
    "gtype" : 6
  }
```


Figura 5.2: Resultado da Importação de Dados com *Script* Python.

A instalação do PostGIS acompanha o *plug-in PostGIS Shapefile and DBF Loader*, que possibilita a importação de arquivos *shapefile* diretamente para o banco de dados. Ele permite algumas configurações como a definição do sistema de referência de coordenadas, a codificação dos caracteres, e a geração de geometrias simples ao invés de multi-partes.

Três conjuntos de dados foram inseridos no Neo4j-Spatial e no PostGIS. Para fins de comparação, a API Java também foi utilizada. Cada conjunto de dados foi inserido no banco de dados cinco vezes através de cada método, tendo seu tempo de importação medido em segundos (s), e a média aritmética gerada a partir dos valores obtidos. O resultado da inserção pode ser observado na Tabela 5.1. Tanto o PostGIS quanto a API Java do Neo4j-Spatial possuem ferramentas desenvolvidas para realizar a importação de arquivos *shapefile* de forma otimizada, apresentando rápido desempenho. O *script* Python, no entanto, elaborado como solução alternativa à implementação original, apresenta perda de desempenho à medida que o número de geometrias armazenadas aumenta.

Tabela 5.1: Comparação do Tempo de Importação de Dados no Neo4j-Spatial e PostGIS.

Nome	**Geometrias**	**Neo4j-Spatial**		**PostGIS**
		Python	Java	
Municípios	247	616,12 (s)	2,68 (s)	0,96 (s)
Rodovias	128	284,78 (s)	0,98 (s)	0,81 (s)
Estados	27	64,44 (s)	0,79 (s)	0,75 (s)

Considerações

Apesar do desempenho similar, a ferramenta de importação de arquivos *shapefile* do Neo4j-Spatial apresenta limitações e não fornece ao usuário a possibilidade de configurações em relação ao arquivo a ser importado como acontece no PostGIS, que possui ferramentas mais consolidadas.

5.1.4 Armazenamento de Dados Geográficos

O Neo4j-Spatial realiza o armazenamento de dados em uma árvore-R, uma estrutura de dados utilizada para indexação de informação geográfica a fim de tornar a pesquisa espacial mais rápida [47]. A árvore-R agrupa tais informações com o uso de uma aproximação retangular chamada caixa envoltória, ou *bounding box*, em que é definido o menor retângulo possível capaz de envolver toda a geometria ali armazenada a partir das coordenadas máximas e mínimas de latitude e longitude. O alto desempenho da consulta espacial se dá pelo fato deste método realizar inicialmente uma consulta mais geral, direcionando a pesquisa primeiro às caixas envoltórias, para então seguir às geometrias nelas contidas [44].

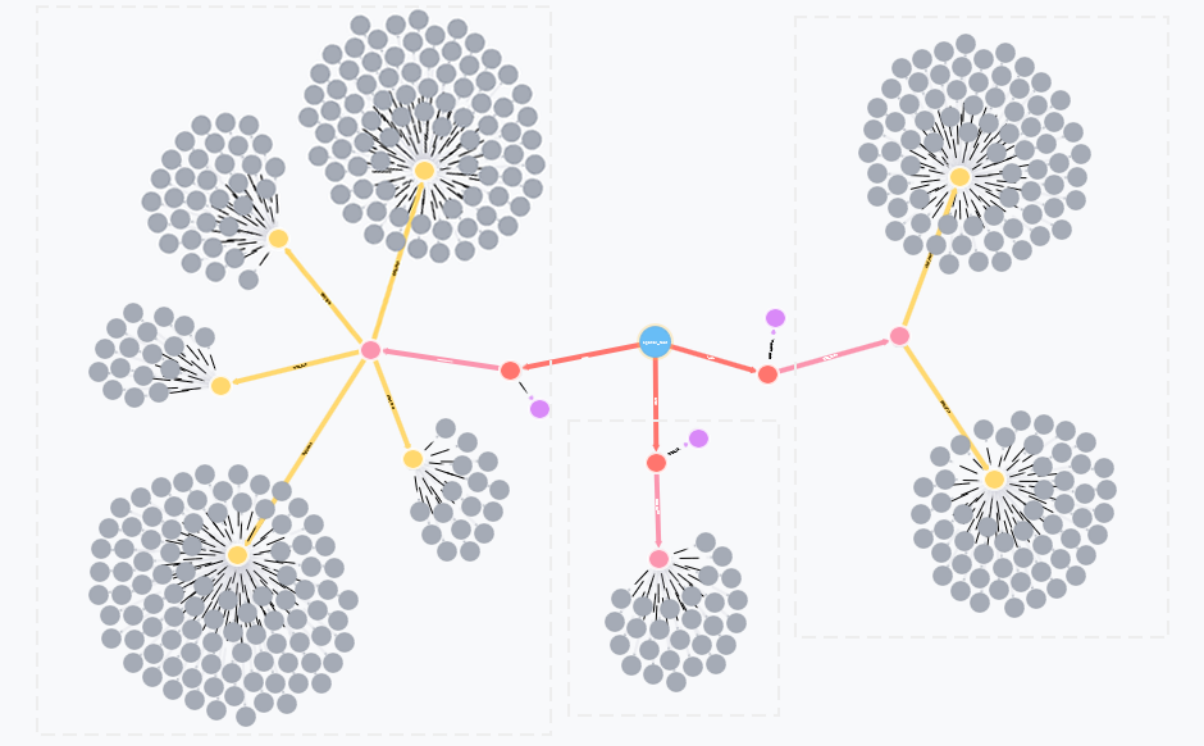


Figura 5.3: Grafo Gerado a Partir da Inserção dos Conjuntos de Dados no Neo4j-Spatial.

A Figura 5.3 representa o grafo gerado no Neo4j-Spatial a partir da importação dos conjuntos de dados. O nó em azul identifica o índice espacial, ponto inicial em todas as operações espaciais. As linhas vermelhas representam as camadas geográficas criadas pela inserção dos três conjuntos de dados armazenados, e os nós de mesma cor contém informações referentes a cada camada como nome, classe utilizada para criação da camada, classe utilizada para criação das geometrias, e a marca temporal da criação da camada. Os

nós em lilás compreendem os metadados, indicando o número de geometrias armazenadas em cada camada. Os nós rosas são as raízes das árvores-R de cada conjunto de dados inserido, e armazenam os valores da *bounding box* geral que envolve todas as geometrias da camada. Os nós amarelos são as *bounding boxes* intermediárias, e por fim, os nós cinzas representam as geometrias individuais, armazenando atributos comuns e propriedades geográficas como a *bounding box* específica e a descrição da própria geometria no formato WKT.

Na Figura 5.4 é apresentada a representação visual dos valores armazenados nas *bounding boxes* existentes no banco de dados em grafo, indicados como os nós rosas e amarelos na Figura 5.3. Dada uma geometria qualquer, ao invés de compará-la com todas as geometrias armazenadas, verifica-se primeiro se sua *bounding box* está contida em uma das *bounding boxes* intermediárias, eliminando as demais opções e diminuindo o número de comparações, e consequentemente otimizando o tempo de busca.

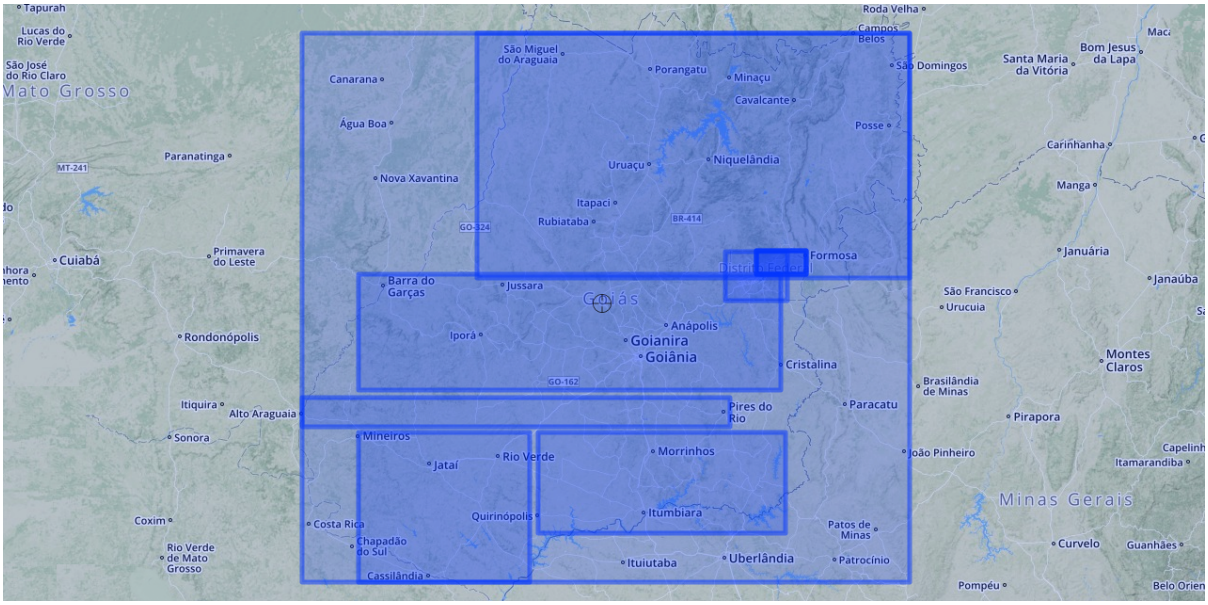


Figura 5.4: Representação das *Bounding Boxes* Intermediárias Armazenadas no Banco.

A inserção de dados através da ferramenta de importação de *shapefile* do PostGIS resulta no armazenamento dos dados em uma tabela similar à qualquer outra tabela do modelo relacional. A principal diferença está no fato desta tabela incluir uma coluna espacial contendo a forma geométrica do dado descrita pelas geometrias definidas pelo OpenGIS. As demais colunas correspondem aos atributos associados ao dado geográfico.

A Figura 5.5 demonstra um trecho da tabela criada com a importação do arquivo *shapefile* contendo dados sobre os municípios brasileiros. A coluna *gid* representa a chave primária da tabela, a coluna *geom* contém a geometria do dado armazenado no formato WKB, indicando ainda a forma geométrica em que ele está representado, e as colunas restantes apresentam os demais atributos relacionados ao dado.

	gid [PK] serial	name character varying(35)	layer character varying(21)	label character varying(35)	codmun1 integer	codmun2 integer	geom geometry(Point)
1	1	Aracaju/SE	Unknown Point Feature	Aracaju/SE	280030	2800308	01010000009097F9BF418942C08E987760B1D225C0
2	2	Belém/PA	Unknown Point Feature	Belém/PA	150140	1501402	01010000006BE9E41F904048C0B88D6877974DF7BF
3	3	Belo Horizonte/MG	Unknown Point Feature	Belo Horizonte/MG	310620	3106200	0101000000463E11006BFA45C0DD60F77F4AD133C0
4	4	Boa Vista/RR	Unknown Point Feature	Boa Vista/RR	140010	1400100	0101000000F2F5F11F34564EC0FBF4B200808E0640
5	5	Brasília/DF	Unknown Point Feature	Brasília/DF	530010	5300108	0101000000D10EF49F17F747C0D72B8E3FA08F2FC0
6	6	Campo Grande/MS	Unknown Point Feature	Campo Grande/MS	500270	5002704	0101000000570CF57FBF524BC05E8DD97F8A7134C0
7	7	Cuiabá/MT	Unknown Point Feature	Cuiabá/MT	510340	5103403	01010000003779B79F790C4CC0187B2FBE68312FC0
8	8	Curitiba/PR	Unknown Point Feature	Curitiba/PR	410690	4106902	0101000000095BCD83F00A348C0DA99E05FB56D39C0
9	9	Florianópolis/SC	Unknown Point Feature	Florianópolis/SC	420540	4205407	01010000007BB84640544648C0A215A35FF9983BC0
10	10	Fortaleza/CE	Unknown Point Feature	Fortaleza/CE	230440	2304400	01010000004431C8BF8B4543C0E1DDB7475CBD0DC0
11	11	Goiânia/GO	Unknown Point Feature	Goiânia/GO	520870	5208707	0101000000 6FC0850091A048C079523900F5AD30C0
12	12	João Pessoa/PB	Unknown Point Feature	João Pessoa/PB	250750	2507507	01010000007073 8CA0806E41C07264471F42761CC0
13	13	Macapá/AP	Unknown Point Feature	Macapá/AP	160030	1600303	0101000000DD683C60808849C0D98385E2F0BEA33F
14	14	Maceió/AL	Unknown Point Feature	Maceió/AL	270430	2704302	01010000000F1416A01EDE41C03827BADF3F5523C0
15	15	Manaus/AM	Unknown Point Feature	Manaus/AM	130260	1302603	0101000000 4D1BFB7F42034EC0C3C12D81D0D108C0
16	16	Natal/RN	Unknown Point Feature	Natal/RN	240810	2408102	0101000000DE60F77FCA9A41C0E6F0E79F912E17C0
17	17	Palmas/TO	Unknown Point Feature	Palmas/TO	172100	1721000	01010000009300E6F8AC2A48C0A036F98FC15524C0
18	18	Porto Alegre/RS	Unknown Point Feature	Porto Alegre/RS	431490	4314902	0101000000DFC10060809D49C0CB85CABF96083EC0
19	19	Porto Velho/RO	Unknown Point Feature	Porto Velho/RO	110020	1100205	0101000000FF543E40C6F34FC09EEAF222628621C0

Figura 5.5: Tabela Gerada a Partir da Importação de Arquivo *Shapefile* no PostGIS.

Para a indexação dos dados, o PostGIS utiliza a árvore-R juntamente com GiST (*Generalized Search Tree* ou árvore de busca genérica), tornando a busca mais robusta. GiST é uma forma generalizada de indexação usada para acelerar buscas em tipos de estruturas de dados irregulares [70].

Considerações

A forma de indexação dos dados geográficos no Neo4j-Spatial está representada de maneira explícita na estrutura em que os dados são armazenados, já apresentando características espaciais devido ao agrupamento dos dados por meio da aproximação retangular das *bounding boxes*. Enquanto no PostGIS o armazenamento gera uma tabela comum e a estrutura de indexação é interna ao banco de dados e transparente para o usuário.

5.1.5 Visualização de Dados Geográficos

Como exposto no Capítulo 2, dados georreferenciados descrevem fenômenos geográficos em que sua localização está associada a uma posição sob a superfície terrestre. É fundamental, portanto, poder visualizar estes dados sob a perspectiva de um mapa.

O Neo4j-Spatial não fornece meios próprios para realizar tal tarefa, mas sua documentação apresenta duas ferramentas, GeoServer[6] e uDig[7], que oferecem suporte ao Neo4j, e permitem a visualização geográfica dos dados armazenados. A seção do uDig contém apenas um *link* redirecionando à Wiki do Neo4j para maiores informações, contudo, a página encontra-se fora do ar. Já a seção dedicada ao GeoServer contém algumas informações gerais a respeito da instalação e da integração com o banco de dados, entretanto, um aviso indica que tais informações foram testadas com a versão 2.1.1 do programa, lançada em junho de 2011.

O Neo4j-Spatial teve atualizações ao longo dos anos juntamente às do Neo4j para permitir que a integração entre as aplicações fosse mantida. Porém, o mesmo não foi realizado com o GeoServer ou uDig, e sua associação encontra-se atualmente comprometida devido à incompatibilidade de bibliotecas.

[6] http://www.geoserver.org/
[7] http://www.udig.refractions.net/

Como abordagem alternativa, foi desenvolvida uma interface *web* com a utilização de HTML, CSS, e do MapBox[8], uma plataforma de desenvolvimento *online* para a criação e a visualização de mapas de forma simples.

Criado no ano de 2011, em resposta à oferta limitada oferecida pelos provedores de mapas como Google Maps, o MapBox tem rapidamente expandido o nicho de mapas customizados [2]. Além de suportar dados provenientes de diversos formatos, incluindo *shapefile*, CSV e GeoJSON, há à disposição ainda o MapBox.js, uma biblioteca *javascript* de código aberto compatível com os navegadores modernos, que permite a inserção de mapas em páginas HTML de forma rápida e fácil [8].

A Figura 5.6 ilustra esta interface, onde os botões verdes funcionam para enquadramento das regiões determinadas, e os botões roxos realizam as operações espaciais selecionadas.

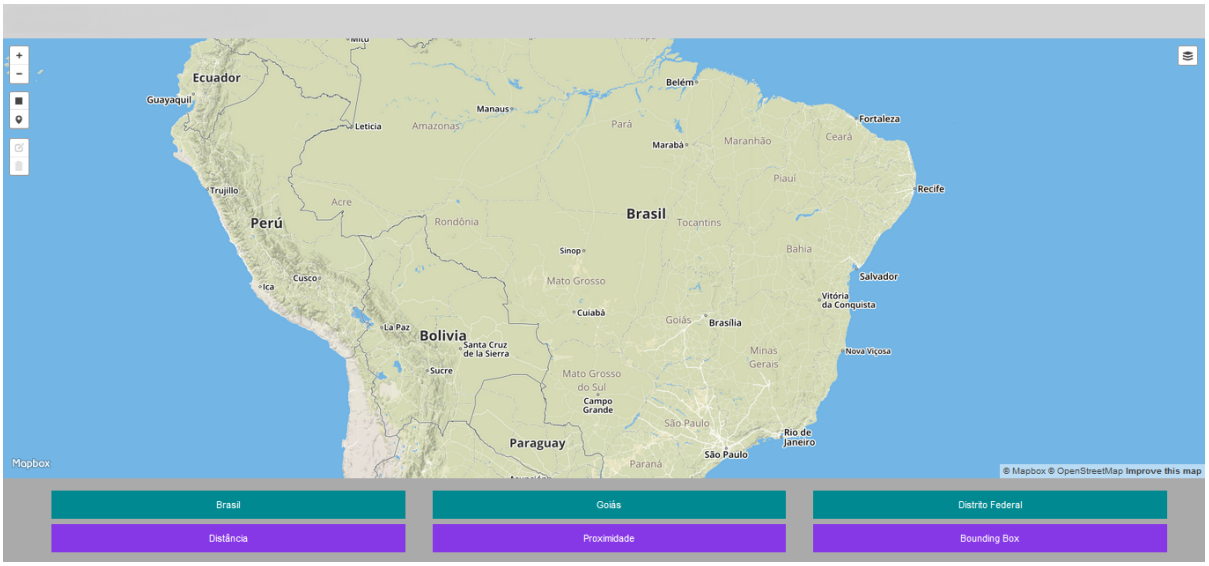


Figura 5.6: Interface Desenvolvida para Visualização dos Dados no Mapa através do MapBox.js.

As Figuras 5.7, 5.8 e 5.9 representam, respectivamente, o resultado da visualização, através da interface desenvolvida, dos dados geográficos do tipo ponto, linha e polígono armazenados no Neo4j.

O PostGIS conta com o suporte a diversas ferramentas, livres e proprietárias, para a visualização de dados geográficos tanto no computador, quanto em servidor. O Quantum GIS[9], mais conhecido como QGIS, é um sistema de informação geográfica de código aberto que roda nos principais sistemas operacionais como Linux, Max OS X e Windows [50], e tende a ser bastante utilizado em conjunto com o PostGIS, principalmente, por iniciantes em SIG devido à sua interface simples e amigável para navegação, edição e reprojeção de mapas [63].

[8] https://www.mapbox.com/
[9] http://www.qgis.org/

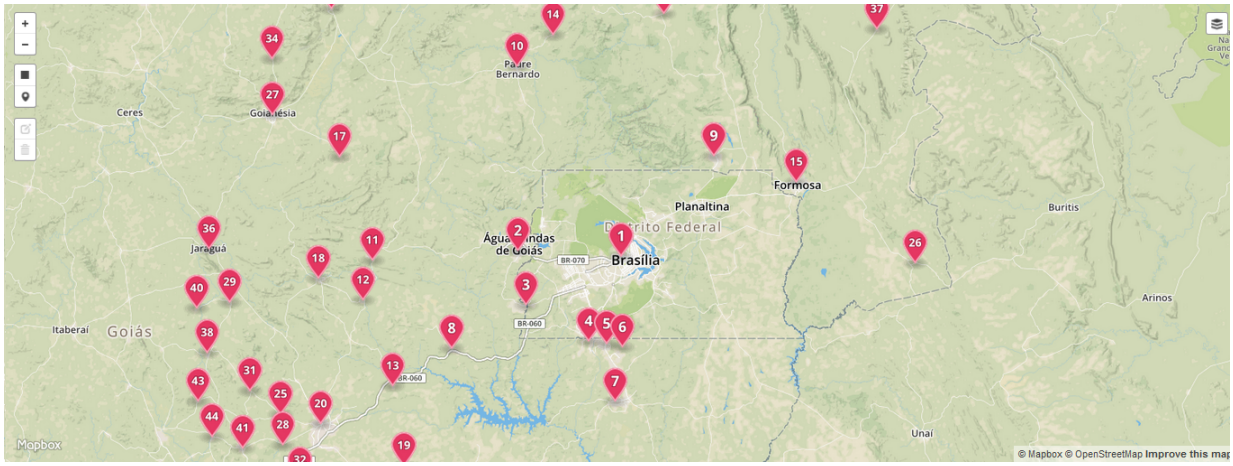


Figura 5.7: Resultado da Visualização de Geometrias do Tipo Ponto no MapBox.js.

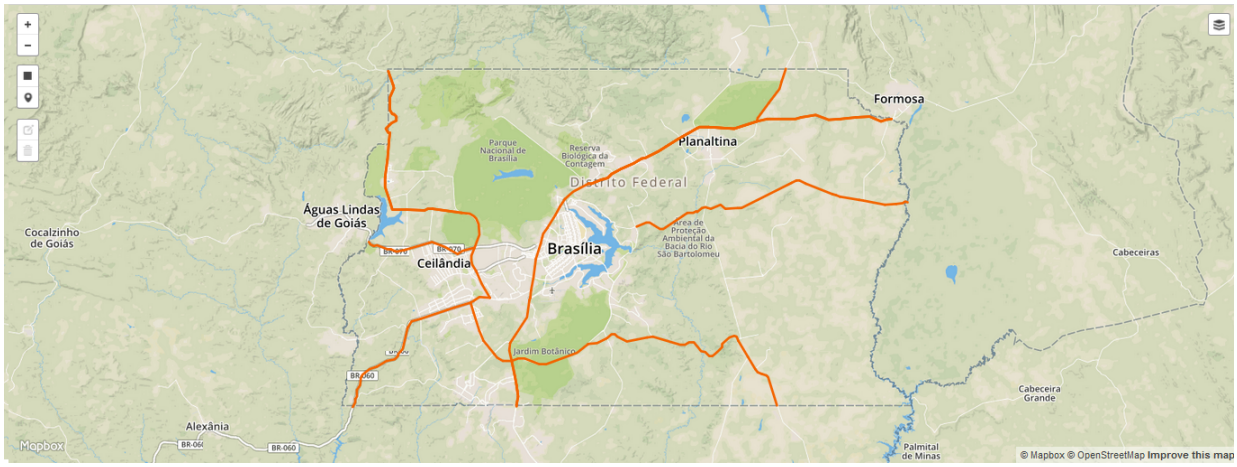


Figura 5.8: Resultado da Visualização de Geometrias do Tipo Linha no MapBox.js.

Figura 5.9: Resultado da Visualização de Geometria do Tipo Polígono no MapBox.js.

A visualização dos dados armazenados no PostgreSQL é realizada pelo QGIS através de uma conexão direta estabelecida com o banco de dados, onde cada tabela se torna uma camada geográfica. A Figura 5.10 ilustra o resultado desta conexão, em que as três tabelas selecionadas aparecem na lateral esquerda, indicando o tipo de geometria utilizado na frente de seus nomes, e a lateral direita mostra a representação visual destas geometrias.

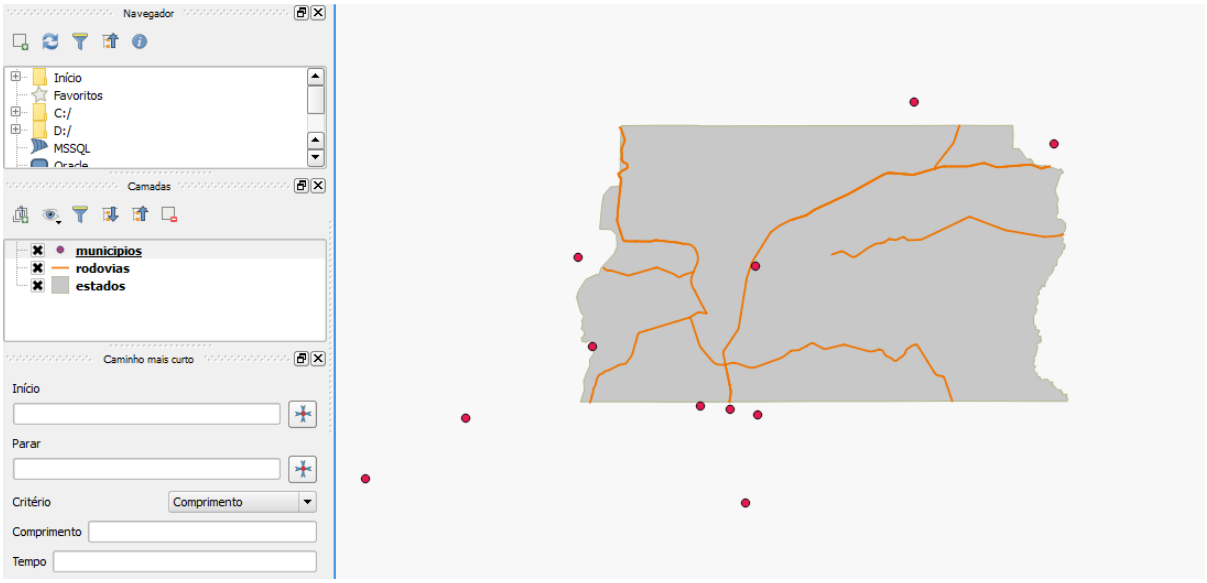


Figura 5.10: Visualização dos Dados Geográficos Armazenados no PostgreSQL através do QGIS.

Porém, a fim de atribuir maior qualidade às análises comparativas, tornando seus resultados mais perceptíveis, foi utilizada para a visualização de dados geográficos do PostGIS a mesma interface desenvolvida para a visualização com o Neo4j, ajustando sua estrutura com a linguagem PHP para efetuar a conexão com o banco de dados e realizar as consultas espaciais.

Considerações

Mais uma vez, por ser um *software* consolidado, o PostgreSQL apresenta vantagens em relação ao Neo4j, principalmente, pela quantidade de ferramentas compatíveis para realizar a visualização dos dados geográficos armazenados e a documentação disponível, facilitando este processo. O Neo4j-Spatial, em contrapartida, apresenta em sua documentação apenas duas ferramentas de visualização de dados geográficos como opção, porém, o suporte de ambas encontra-se defasado, não sendo possível a integração com a versão mais atual do Neo4j.

A visualização através da interface desenvolvida, no entanto, mostrou-se simples com ambos os bancos de dados, pois o Mapbox.js utiliza o formato GeoJSON para a representação no mapa, e tanto o Neo4j-Spatial quanto o PostGIS possibilitam o resultado de suas consultas nesse formato.

5.1.6 Consultas Espaciais

Como abordado no Capítulo 2, bancos de dados são sistemas computadorizados de armazenamento em que usuários podem realizar operações básicas como inserir, recuperar, atualizar e deletar dados [31]. Nesta seção são apresentadas três operações espaciais realizadas para a recuperação de dados geográficos dos bancos. Para cada caso são definidas as consultas referentes ao Neo4j-Spatial em REST, e ao PostGIS em SQL.

A Wiki do Neo4j-Spatial provê documentação no que diz respeito às operações suportadas via REST, dispondo dos comandos necessários a sua execução, descrição de cada atributo a ser enviado, e exemplo de resposta a ser esperada do servidor. O mesmo pode ser dito em relação ao PostGIS, que apresenta documentação[10] extensa e bem detalhada, cobrindo as funções disponíveis para manipulação de dados geográficos.

Para análise entre os bancos de dados, foram utilizadas as consultas apresentadas na Tabela 5.2, onde são listadas as funções espaciais de distância, proximidade e *bounding box* e suas descrições.

Tabela 5.2: Funções Espaciais Realizadas para Análise entre os Bancos de Dados.

Função	**Descrição**
Distância	Procura por geometrias que se encontram dentro de uma determinada distância do ponto de interesse
Proximidade	Procura pela geometria mais próxima do ponto de interesse dentro de uma determinada distância
Bounding Box	Procura por geometrias contidas em uma *bounding box*

Distância

Esta consulta tem como objetivo encontrar as geometrias que se encontram a um determinado raio de distância de um dado ponto. Foram definidos como parâmetros para ambas as ferramentas a distância de 100km e o ponto equivalente à Brasilia de acordo com as coordenadas fornecidas pelo IBGE.

O *Script* 5.2 apresenta como essa consulta é feita com o Neo4j-Spatial. É enviado uma chamada *POST* ao endereço *url* da operação, juntamente com o cabeçalho definindo a codificação e o formato dos dados e os parâmetros como um conjunto de chave-valor. A consulta recebe como parâmetros o nome da camada trabalhada, o ponto de partida definido por sua latitude e longitude, e a distância deste ponto em quilômetros na qual a pesquisa será efetuada.

A resposta do servidor é recebida no formato JSON e convertida para GeoJSON através da interface desenvolvida para sua visualização. O resultado da consulta pode ser observado na Figura 5.11, em que foram retornados dezesseis municípios a 100km de Brasília, ordenados de acordo com a distância.

[10] http://www.postgis.net/docs/

```
POST http://localhost:7474/db/data/ext/SpatialPlugin/graphdb/
    findGeometriesWithinDistance

Accept: application/json; charset=UTF-8
Content-Type: application/json

{
  "layer" : "municipios",
  "pointX" : -47.9304084722222,
  "pointY" : -15.7805194722222,
  "distanceInKm" : 100
}
```

Script 5.2: Consulta em Distância Realizada no Neo4j-Spatial Utilizando REST.

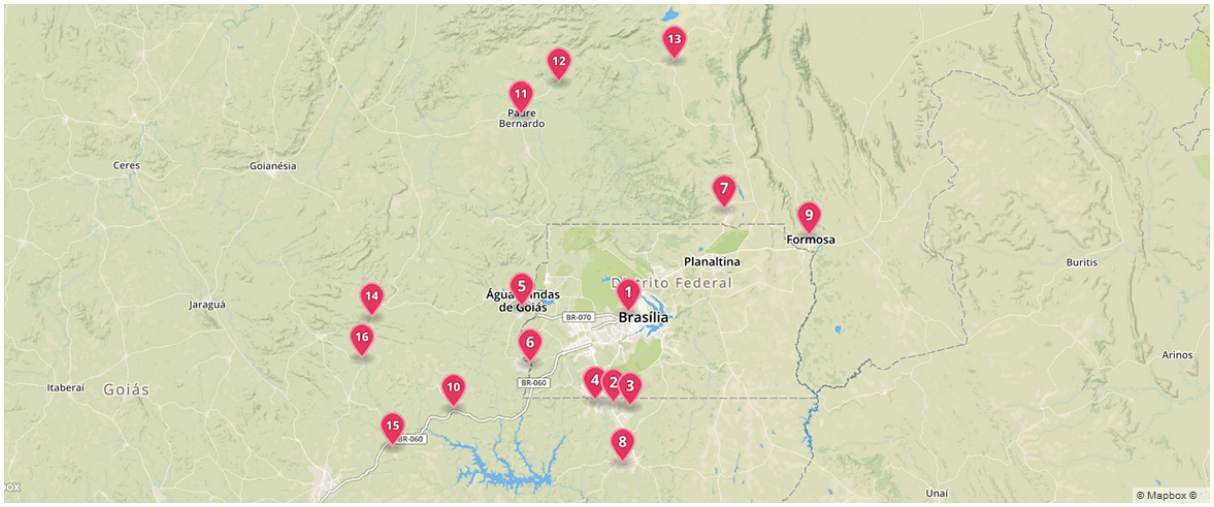


Figura 5.11: Resultado da Consulta em Distância Utilizando o Neo4j-Spatial.

No PostGIS a consulta é realizada de forma semelhante, recebendo como parâmetros o nome da tabela, o ponto de referência, e a distância em metros. Foi utilizada a função **ST Distance Sphere**, que retorna a distância linear em metros entre dois pontos, e a função **ST MakePoint**, que cria um ponto a partir de sua latitude e longitude. O *Script* 5.3 representa esta busca em SQL, onde são selecionados o id, o nome, e a geometria da tabela "municipios" cuja distância de Brasília seja menor ou igual a cem quilômetros. A geometria foi selecionada no formato GeoJSON pela função **ST AsGeoJSON** para facilitar a visualização dos resultados obtidos, demonstrados na Figura 5.12. Como pode ser observado, também foram retornados dezesseis municípios dentro da distância de 100km de Brasília, ordenados, porém, em ordem alfabética.

```
SELECT gid, name, ST_AsGeoJSON(geom)
FROM municipios
WHERE ST_Distance_Sphere(geom, ST_MakePoint(-47.9304084722222,
    -15.7805194722222)) <= 100000
```

Script 5.3: Consulta em Distância Realizada no PostGIS Utilizando SQL.

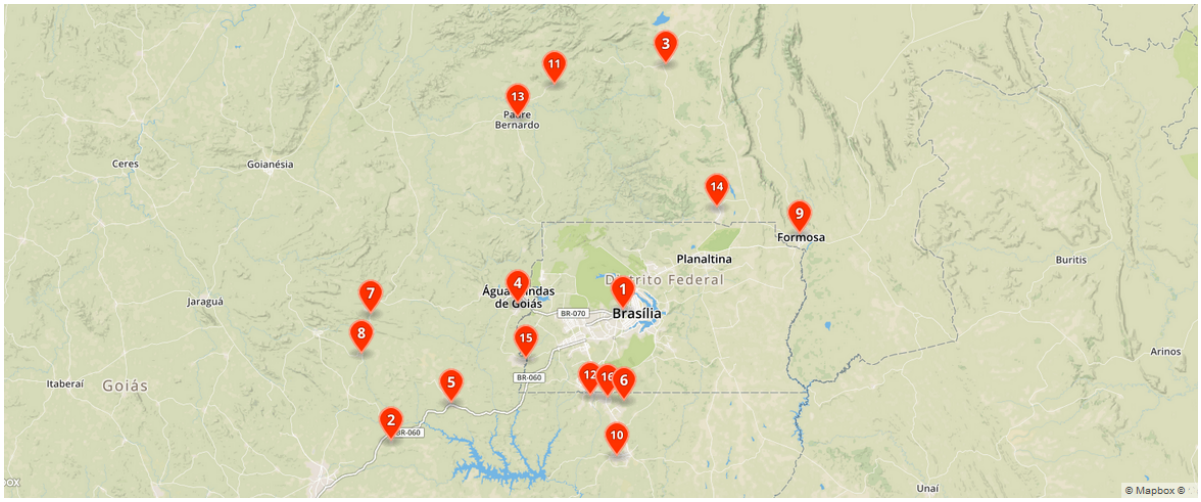


Figura 5.12: Resultado da Consulta em Distância Utilizando o PostGIS.

Proximidade

Esta consulta tem como objetivo encontrar a geometria mais próxima a um determinado ponto dentro de um raio de distância. Foram definidos como parâmetros a distância de 100km e o ponto referente à Brasilia, segundo as coordenadas providas pelo IBGE.

O Neo4j-Spatial realiza essa consulta como representado no *Script* 5.4, que possui a mesma estrutura que a consulta em distância. A chamada *POST* retorna do servidor um vetor contendo as geometrias encontradas dentro da distância determinada, ordenadas de forma crescente em relação ao ponto definido, indo da geometria mais próxima à mais distante. Portanto, a fim de determinar o ponto mais próximo de Brasília, foi selecionada a segunda posição deste vetor, uma vez que a primeira refere-se ao próprio ponto de partida. O resultado da consulta é apresentado na Figura 5.13, onde Valparaíso de Goiás é dado como o município mais próximo de Brasília.

```
POST http://localhost:7474/db/data/ext/SpatialPlugin/graphdb/
    findClosestGeometries

Accept: application/json; charset=UTF-8
Content-Type: application/json

{
  "layer" : "municipios",
  "pointX" : -47.9304084722222,
  "pointY" : -15.7805194722222,
  "distanceInKm" : 100
}
```

Script 5.4: Consulta em Proximidade Realizada no Neo4j-Spatial Utilizando REST.

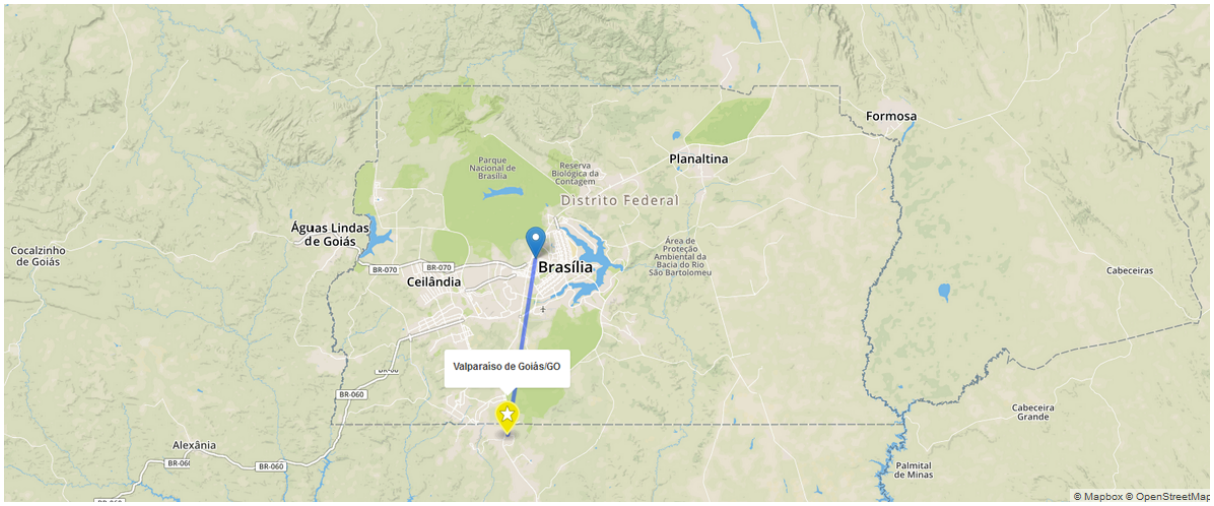


Figura 5.13: Resultado da Consulta em Proximidade Utilizando o Neo4j-Spatial.

O *Script* 5.5 demonstra a consulta feita com o PostGIS utilizando as funções **ST MakePoint** para a criação do ponto a partir de suas coordenadas, **ST DWithin** que filtra as geometrias a determinada distância do ponto de referência, e **ST Distance** que retorna a distância em metros entre dois pontos.

```
SELECT gid, name, ST_AsGeoJSON(geom)
FROM municipios
WHERE ST_DWithin(geom, ST_MakePoint(-47.9304084722222, -15.7805194722222)
    , 100000)
ORDER BY ST_Distance(geom, ST_MakePoint(-47.9304084722222,
    -15.7805194722222))
LIMIT 2
```

Script 5.5: Consulta em Proximidade Realizada no PostGIS Utilizando SQL.

A consulta seleciona os atributos id, nome e geometria da tabela "municipios" dentro do raio de cem quilômetros do ponto de referência, ordenando os resultados pela sua distância, e filtrando apenas os dois primeiros, referentes à Brasília e ao município mais próximo. A Figura 5.14 apresenta o resultado desta consulta, que também retornou Valparaíso de Goiás como o município mais próximo.

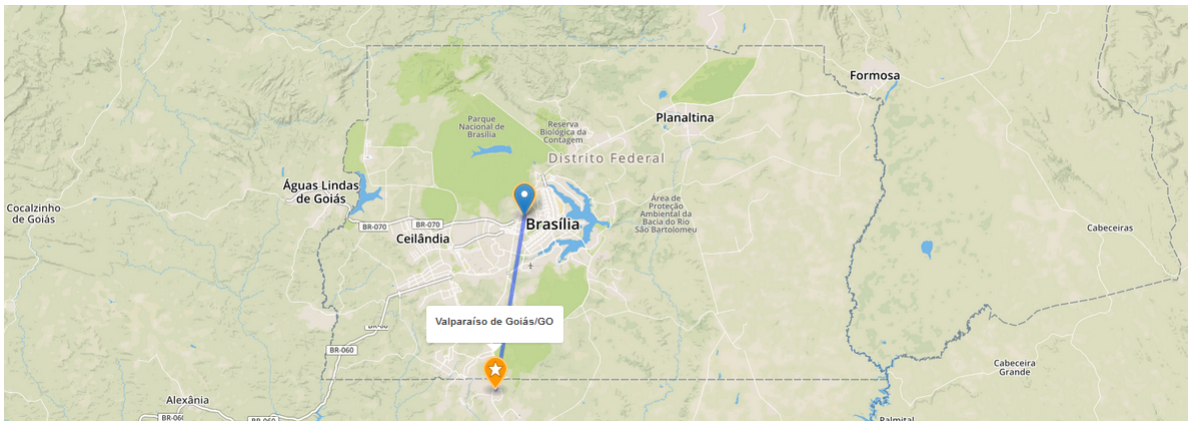


Figura 5.14: Resultado da Consulta em Proximidade Utilizando o PostGIS.

Bounding Box

Esta consulta tem como objetivo retornar as geometrias contidas em uma *bounding box*. Foram definidos como parâmetros os valores máximos e mínimos de latitude e longitude, de tal forma que a *bounding box* criada envolvesse o Distrito Federal e entorno.

A consulta por geometrias dentro de uma *bounding box* no Neo4j-Spatial recebe estes valores e o nome da camada a ser pesquisada como parâmetros. O *Script* 5.6 representa a consulta realizada, que segue o mesmo procedimento das anteriores. O resultado retornado é um vetor contendo oito geometrias, que após convertidas para GeoJSON podem ser observadas na Figura 5.15.

```
POST http://localhost:7474/db/data/ext/SpatialPlugin/graphdb/
    findGeometriesInBBox
Accept: application/json; charset=UTF-8
Content-Type: application/json

{
  "layer" : "municipios",
  "minx" : -48.339843750025791, "maxx" : -47.248077392578125,
  "miny" : -16.105195094166007, "maxy" : -15.444414528150116
}
```

Script 5.6: *Script* REST Utilizado para a Consulta em *Bounding Box* no Neo4j-Spatial.

Figura 5.15: Resultado da Consulta em *Bounding Box* Utilizando o Neo4j-Spatial.

No PostGIS também são passados como parâmetros apenas o nome da tabela e os pontos que delimitam a *bounding box*, que será construída pela função **ST MakeEnvelope** a partir destes pontos. O operador "@" na cláusula *WHERE* filtra a busca selecionando as geometrias contidas pela *bounding box*. O *Script* 5.7 expressa tal consulta em SQL e a Figura 5.16 ilustra a representação visual dos resultados obtidos, que também apresentou oito geometrias dentro da *bounding box* determinada.

```
SELECT gid, name, ST_AsGeoJSON(geom)
FROM municipios
WHERE geom @ ST_MakeEnvelope(-48.339843750025791, -16.105195094166007,
    -47.248077392578125, -15.444414528150116);
```

Script 5.7: *Script* SQL Utilizado para Realizar a Consulta em *Bounding Box* no PostGIS.

Figura 5.16: Resultado da Consulta em *Bounding Box* Utilizando o PostGIS.

Considerações

Como pôde ser observado, foi possível realizar todas as consultas espaciais propostas em ambos os bancos de dados. O resultado destas consultas se mostrou o mesmo, tanto no Neo4j-Spatial quanto no PostGIS dentro de uma mesma interface, e não apresentou uma diferença significativa de desempenho. Entretanto, por ser um *software* mais consolidado, o PostGIS possui ferramentas melhor otimizadas para manipulação de dados geográficos, e o uso do SQL como linguagem falicita a realização de consultas espaciais.

5.1.7 Suporte e Usabilidade

Apesar do surgimento e expansão de novas abordagens para o gerenciamento de bancos de dados, o modelo relacional mantém sua hegemonia em relação aos outros sistemas, e o uso do SQL como linguagem unificada favorece este fato. O PostgreSQL e sua extensão espacial PostGIS possuem suporte comercial através de empresas que fornecem serviço profissional, e amplo suporte comunitário por meio de fórum *online*, grupos de discussão, listas de *e-mail* e seção de dúvidas frequentes. Além disso, dispõem de uma Wiki própria e a documentação é extensa e mantida atualizada. Diversos tutoriais e exemplos são facilmente encontrados na Internet, tornando mais rápida a configuração do banco de dados.

Os bancos de dados NoSQL são relativamente novos, e apesar de estarem ganhando cada vez mais espaço em aplicações diversas, sua adoção ainda é focada na resolução de problemas específicos. O modelo orientado a grafos, que apresentou o maior crescimento em adoção nos últimos anos, sofre por não ter uma linguagem única, fazendo com que o suporte dado à uma implementação dependa do *software* utilizado em questão.

O Neo4j fornece suporte técnico principalmente através de seu *site*, por meio de grupos de discussão (onde a comunidade de usuários, ainda que pequena, está sempre disposta a solucionar dúvidas), fórum e listas de *e-mail*. Há ainda uma Wiki contendo informações gerais e a própria documentação do programa. Esta, porém, nem sempre encontra-se completa ou atualizada.

Se já não existem na Internet muitas referências de implementação utilizando o Neo4j, referências com sua extensão espacial, o Neo4j-Spatial, existem menos ainda, e sua pouca — e desatualizada — documentação dificulta o trabalho de quem pretende configurar um banco de dados geográfico orientado a grafos, tornando o processo dispendioso.

5.2 Conclusão

A Tabela 5.3 apresenta um resumo das análises realizadas, comparando os aspectos referentes à funcionalidade, inserção, armazenamento e visualização de dados geográficos, consultas espaciais, suporte e usabilidade do sistema.

Tabela 5.3: Comparação entre os Aspectos Analisados no Neo4j-Spatial e no PostGIS.

Nome	**Neo4j-Spatial**	**PostGIS**
Funcionalidades	suporte à funções espaciais limitado	amplo conjunto de funcionalidades espaciais
Inserção	ferramenta para importação de *shapefile* apresenta erros	ferramenta otimizada que oferece opções de configuração para a importação
Armazenamento	estrutura de indexação explícita no armazenamento de dados	indexação interna e transparente ao usuário
Visualização	versão atual não é suportada pelas ferramentas listadas na documentação	grande quantidade de ferramentas compatíveis
Consultas Espaciais	o uso de REST simplifica as consultas	o uso do SQL facilita a realização das consultas
Suporte	fornecido principalmente pela comunidade de desenvolvedores	suporte especializado e comunitário

Embora o Neo4j tenha sido eleito por Batista et al. em [33] como a melhor alternativa NoSQL para o armazenamento e a manipulação de dados geográficos, a sua extensão espacial Neo4j-Spatial apresentou desvantagens na maioria dos apectos abordados durante a avaliação comparativa realizada. Por outro lado, o PostgreSQL mostrou-se um banco de dados bem estabelecido, e com uma extensão espacial, o PostGIS, que possui ferramentas mais consolidadas para a manipulação de dados geográficos, bem como um amplo catálogo de funções espaciais.

Capítulo 6

Conclusões e Trabalhos Futuros

Neste trabalho foi realizado um estudo acerca do suporte aos dados geográficos de um banco de dados orientado a grafo em relação a um banco de dados baseado no modelo relacional a partir de fatores como o armazenamento, a inserção e a visualização de dados geográficos, o desempenho das consultas espaciais e a usabilidade do sistema. Para tal, foi desenvolvida uma interface *web* que possibilitasse a integração com os dois bancos de dados.

Previamente à implementação, foram introduzidos os sistemas de informação geográfica — abordando conceitos básicos relacionados, sua evolução, estrutura e principais funcionalidades — e os bancos de dados, apresentando a evolução desses sistemas para oferecer suporte aos dados geográficos e sua dificuldade para lidar com o chamado *Big Data*. Em seguida, foram abordados os bancos de dados NoSQL — que surgiram como alternativa ao modelo relacional para o armazenamento de grandes quantidades de dados mais complexos e menos estruturados — apresentando suas principais categorias: chave/valor, documento, coluna e grafo.

Dentre as ferramentas NoSQL avaliadas, o banco de dados orientado a grafo Neo4j demonstrou possuir o melhor conjunto de funcionalidades para a manipulação e o armazenamento de dados geográficos e, portanto, foi selecionado para a implementação.

Os resultados das análises demonstram que, para as condições aplicadas, a extensão espacial Neo4j-Spatial apresenta desvantagens na maioria dos apectos abordados durante a avaliação comparativa realizada, devido, principalmente, ao fato de ser um sistema relativamente novo e não estar bem consolidado como o PostgreSQL. Apesar disso, as consultas espaciais no Neo4j apresentaram desempenho similar às do modelo relacional dentro de uma mesma interface.

As principais vantagens da utilização do Neo4j estão baseadas em suas premissas de sistema NoSQL como a escalabilidade horizontal e o melhor gerenciamento de dados semi e/ou não estruturados. Entretanto, em relação à manipulação e ao armazenamento de dados geográficos, o seu pequeno catálogo de funções espaciais pode representar limitações em aplicações que necessitam de análises mais complexas.

6.1 Trabalhos Futuros

Algumas sugestões de melhorias a serem implementadas em trabalhos futuros são:

- Além do *shapefile*, trabalhar com alguns dos diferentes tipos de arquivos de dados geográficos apresentados na Seção 4.2;
- Uma das principais características dos modelos NoSQL é a escalabilidade horizontal. Pode-se, portanto, trabalhar com dados distribuídos entre mais máquinas;
- Tendo em vista que o Neo4j suporta o armazenamento de bilhões de nós e destaca-se em situações em que estes nós estão altamente conectados, pode-se trabalhar com um conjunto maior de dados e ampliar a conectividade entre eles. Além disso, pode-se trabalhar com a API Java para realizar consultas espaciais mais complexas a partir destes dados, como união, disjunção e interseção de geometrias, dentre outras funções espaciais suportadas.

Referências

[1] Neo4j spatial. http://neo4j-contrib.github.io/spatial/, 2013. (Acessado em junho de 2016). 37

[2] The new cartographers. http://www.washingtonpost.com/sf/brand-connect/wp/2013/07/22/the-new-cartographers/, 2013. (Acessado em abril de 2016). 46

[3] Raster basics. http://desktop.arcgis.com/en/arcmap/10.3/manage-data/geodatabases/raster-basics.htm, 2014. (Acessado em junho de 2016). 12

[4] Cypher query language. http://neo4j.com/docs/stable/cypher-query-lang.html, 2015. (Acessado em outubro de 2015). 29

[5] Geospatial indexes and queries. https://docs.mongodb.org/manual/applications/geospatial-indexes/, 2015. (Acessado em dezembro de 2015). 37

[6] Introducing spatial views for location aware applications with couchbase server 4.0. http://www.couchbase.com/nosql-resources/presentations/introducing-spatial-views-for-location-aware-applications-with-couchbase-server-4.0.html, 2015. (Acessado em junho de 2016). 36

[7] Titan documentation. http://s3.thinkaurelius.com/docs/titan/1.0.0/benefits.html, 2015. (Acessado em junho de 2016). 27, 37

[8] Api documentation. https://www.mapbox.com/mapbox.js/, 2016. (Acessado em abril de 2016). 46

[9] Dbms popularity broken down by database model. http://db-engines.com/en/ranking_categories, 2016. (Acessado em junho de 2016). 30, 31

[10] Lucene spatial. http://orientdb.com/docs/2.2/Spatial-Index.html, 2016. (Acessado em junho de 2016). 37

[11] Orientdb. http://orientdb.com/orientdb/, 2016. (Acessado em junho de 2016). 27

[12] Daniel J Abadi, Peter A Boncz, and Stavros Harizopoulos. Column-oriented database systems. *Proceedings of the VLDB Endowment*, 2(2):1664–1665, 2009. 22

[13] Ablimit Aji, Fusheng Wang, Hoang Vo, Rubao Lee, Qiaoling Liu, Xiaodong Zhang, and Joel Saltz. Hadoop gis: a high performance spatial data warehousing system over mapreduce. *Proceedings of the VLDB Endowment*, 6(11):1009–1020, 2013. 32

[14] Paul Andlinger. Rdbms dominate the database market, but nosql systems are catching up. http://db-engines.com/en/blog_post/23, 2013. (Acessado em outubro de 2015). 30, 31

[15] Renzo Angles and Claudio Gutierrez. Survey of graph database models. *ACM Computing Surveys (CSUR)*, 40(1):1, 2008. 26

[16] Stan Aronoff. Geographic information systems: a management perspective. 1989. 5

[17] Gene Bellinger, Durval Castro, and Anthony Mills. Data, information, knowledge, and wisdom. 2004. 4

[18] Tor Bernhardsen. *Geographic information systems: an introduction*. John Wiley & Sons, 2002. 5

[19] John Adrian Bondy and Uppaluri Siva Ramachandra Murty. *Graph theory with applications*, volume 290. Macmillan London, 1976. 24

[20] Eric A. Brewer. Towards robust distributed systems. http://www.cs.berkeley.edu/ brewer/cs262b-2004/PODC-keynote.pdf, 2000. ACM Symposium on Principles of Distributed Computing (PODC). 18

[21] Craig Brown. Cap theorem diagram for distribution. http://blog.nosqltips.com/2011/04/cap-diagram-for-distribution.html, 2011. (Acessado em setembro de 2015). 17

[22] Peter A Burrough. Principles of geographical information systems for land resources assessment. 1986. 5

[23] Gilberto Câmara. Representação computacional de dados geográficos. *CASANOVA, MA et al. Banco de dados geográficos. Curitiba: Mundogeo*, pages 11–52, 2005. 14

[24] Marco Antonio Casanova, Gilberto Camara, Clodoveu A Davis Jr, Lúbia Vinhas, and GR de Queiroz. *Bancos de dados geográficos*. MundoGEO Curitiba, 2005. 7, 12

[25] Rick Cattell. Scalable sql and nosql data stores. *ACM SIGMOD Record*, 39(4):12–27, 2011. 1, 16, 19, 26

[26] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C Hsieh, Deborah A Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E Gruber. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26(2):4, 2008. 16, 22

[27] Chris Clifton and Hector Garcie-Molina. The design of a document database. In *Proceedings of the ACM conference on Document processing systems*, pages 125–134. ACM, 2000. vii, 21

[28] J Terry Coppock and David W Rhind. The history of gis. *Geographical information systems: Principles and applications*, 1(1):21–43, 1991. 5

[29] David J Cowen. Gis versus cad versus dbms: What are the differences?. *Photogramm. Eng. Remote Sens.*, 54(11):1551–1555, 1988. 5

[30] Jack Dangermond. *A classification of software components commonly used in geographic information systems*, volume 3051. Taylor & Francis: London, 1990. 5

[31] Christopher J Date. *Introdução a sistemas de bancos de dados*. Elsevier Brasil, 2004. 9, 49

[32] Gilberto Ribeiro de Queiroz, Antônio Miguel Vieira Monteiro, and Gilberto Câmara. Bancos de dados geográficos e sistemas nosql: onde estamos e para onde vamos. *Revista Brasileira de Cartografia*, 65(3), 2013. 38

[33] Cláudio de Souza Baptista, Carlos Eduardo Santos Pires, Daniel Farias Batista Leite, and Maxwell Guimarães de Oliveira. Nosql geographic databases: an overview. *Geographical Information Systems: Trends and Technologies*, page 73, 2014. 33, 38, 55

[34] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: amazon's highly available key-value store. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 205–220. ACM, 2007. 16, 20

[35] Kenneth J Dueker. Geographic information systems and computer-aided mapping. *Journal of the American Planning Association*, 53(3):383–390, 1987. 5

[36] Ramez Elmasri, Shamkant B Navathe, Marília Guimarães Pinheiro, Claudio Cesar Canhette, Glenda Cristina Valim Melo, Claudia Vicci Amadeu, and Rinaldo de Oliveira Morais. *Sistemas de banco de dados*. Pearson Addison Wesley, 2005. 9, 10

[37] ESRI ESRI. Shapefile technical description. *An ESRI White Paper*, 1998. 39

[38] Stewart Fotheringham and Peter Rogerson. *Spatial analysis and GIS*. CRC Press, 2013. 7

[39] Andrew U Frank. Requirements for a database management system for a gis. *Photogramm. Eng. Remote Sens.*, 54(11):1557–1564, 1988. 9

[40] Hector Garcia-Molina, Jeffrey D Ullman, and Jennifer Widom. *Database system implementation*, volume 654. Prentice Hall Upper Saddle River, NJ:, 2000. 10

[41] Seth Gilbert and Nancy Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *ACM SIGACT News*, 33(2):51–59, 2002. 17

[42] Michael F Goodchild. Geographic information systems. *Journal of Retailing*, 67(1):3–15, 1991. 7

[43] Michael F Goodchild and Karen Kathleen Kemp. *NCGIA Core Curriculum: Introduction to GIS*, volume 1. National Center for Geographic Information and Analysis, University of California at Santa Barbara, 1990. 11

[44] Michael F Goodchild, Paul A Longley, David J Maguire, and David W Rhind. *Geographic information systems and science*, volume 2. John Wiley and Sons, Chichester, 2005. 4, 12, 14, 43

[45] Mark Graves, Ellen R Bergeman, and Charles B Lawrence. A graph-theoretic data model for genome mapping databases. In *System Sciences, 1995. Proceedings of the Twenty-Eighth Hawaii International Conference on*, volume 5, pages 32–41. IEEE, 1995. 26

[46] Ralf Hartmut Güting. An introduction to spatial database systems. *The VLDB Journal—The International Journal on Very Large Data Bases*, 3(4):357–399, 1994. 11

[47] Antonin Guttman. *R-trees: a dynamic index structure for spatial searching*, volume 14. ACM, 1984. 34, 43

[48] Carlos Alberto Heuser. *Projeto de banco de dados*. Sagra Luzzatto, 2001. 18

[49] Jonathan Hey. The data, information, knowledge, wisdom chain: the metaphorical link. *Intergovernmental Oceanographic Commission*, 2004. 4

[50] Marco Hugentobler. Quantum gis. In *Encyclopedia of GIS*, pages 935–939. Springer, 2008. 46

[51] Michael D Kennedy. *Introducing geographic information systems with ARCGIS: a workbook approach to learning GIS*. John Wiley & Sons, 2013. 5, 8

[52] Neal Leavitt. Will nosql databases live up to their promise? *Computer*, 43(2):12–14, 2010. 17

[53] Jugurta Lisboa Filho. Projeto de banco de dados para sistemas de informação geográfica. *SBC-Revista Eletrônica de Iniciação Científica*, 2001. 1, 5, 12, 13

[54] Olli Luukkonen, Keijo Heljanko, et al. Survey of nosql database engines for big data. 2015. 20

[55] David J Maguire. An overview and definition of gis. *Geographical information systems: Principles and applications*, 1:9–20, 1991. 5, 6

[56] David Martin. *Geographic information systems: socioeconomic applications*. Psychology Press, 1996. 1, 5, 6

[57] Andrew McAfee, Erik Brynjolfsson, Thomas H Davenport, DJ Patil, and Dominic Barton. Big data. *The management revolution. Harvard Bus Rev*, 90(10):61–67, 2012. 16

[58] Christopher McCarthy. Does nosql have a place in gis?-an open-source spatial database performance comparison with proven rdbms. 2014. 37

[59] Dan McCreary and Ann Kelly. Making sense of nosql. *Greenwich, Conn.: Manning Publications*, 2013. 1, 20, 21, 27

[60] Justin J Miller. Graph database applications and concepts with neo4j. In *Proceedings of the Southern Association for Information Systems Conference, Atlanta, GA, USA March 23rd-24th*, 2013. 26

[61] ABM Moniruzzaman and Syed Akhter Hossain. Nosql database: New era of databases for big data analytics-classification, characteristics and comparison. *arXiv preprint arXiv:1307.0191*, 2013. 1, 16, 20, 27

[62] Ameya Nayak, Anil Poriya, and Dikshay Poojary. Type of nosql databases and its comparison with relational databases. *International Journal of Applied Information Systems*, 5(4):16–19, 2013. 21

[63] Regina O Obe and Leo S Hsu. *PostGIS in action*. Manning Publications Co., 2015. 13, 15, 39, 46

[64] Onofrio Panzarino. *Learning Cypher*. Packt Publishing Ltd, 2014. 29

[65] Cesare Pautasso, Olaf Zimmermann, and Frank Leymann. Restful web services vs. big'web services: making the right architectural decision. In *Proceedings of the 17th international conference on World Wide Web*, pages 805–814. ACM, 2008. 40

[66] Hasso Plattner. A common database approach for oltp and olap using an in-memory column database. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 1–2. ACM, 2009. 22

[67] Jaroslav Pokorny. Nosql databases: a step to database scalability in web environment. *International Journal of Web Information Systems*, 9(1):69–82, 2013. 17, 18

[68] Dan Pritchett. Base: An acid alternative. *Queue*, 6(3):48–55, 2008. 18, 19

[69] Raghu Ramakrishnan and Johannes Gehrke. Database management systems. 2000. 10

[70] Paul Ramsey et al. Postgis manual. *Refractions Research Inc*, 2005. 40, 45

[71] Ian Robinson, Jim Webber, and Emil Eifrem. *Graph databases*. "O'Reilly Media, Inc.", 2013. 26

[72] Marko A Rodriguez and Peter Neubauer. Constructions from dots and lines. *Bulletin of the American Society for Information Science and Technology*, 36(6):35–41, 2010. vii, 24, 25

[73] A de Rolf. *Principles of geographic information systems: an introductory textbook*, volume 1. International Institute for Geo-Information Science and Earth Observation, 2004. 11

[74] Johannes Scholz. *Coping with Dynamic, Unstructured Data Sets–NoSQL a Buzzword or a Savior?* na, 2011. 33

[75] Valdemar W Setzer. Dado, informação, conhecimento e competência. *DataGrama-Zero Revista de Ciência da Informação, n. 0*, 1999. 4

[76] Christof Strauch, Ultra-Large Scale Sites, and Walter Kriha. Nosql databases. *Lecture Notes, Stuttgart Media University*, 2011. 17, 21

[77] Claudio Tesoriero. *Getting Started with OrientDB*. Packt Publishing Ltd, 2013. 27

[78] Krishnaiyan Thulasiraman and Madisetti NS Swamy. *Graphs: theory and algorithms*. John Wiley & Sons, 2011. 24

[79] Shashank Tiwari. *Professional NoSQL*. John Wiley & Sons, 2011. 20

[80] Gaurav Vaish. *Getting started with NoSQL*. Packt Publishing Ltd, 2013. 21

[81] Rik Van Bruggen. *Learning Neo4j*. Packt Publishing Ltd, 2014. 21, 23, 24, 26, 28, 29

[82] Jim Webber. A programmatic introduction to neo4j. In *Proceedings of the 3rd annual conference on Systems, Programming, and Applications: Software for Humanity*, pages 217–218. ACM, 2012. 28

[83] Douglas Brent West et al. *Introduction to graph theory*, volume 2. Prentice hall Upper Saddle River, 2001. vii, 23

[84] Michael F Worboys and Matt Duckham. *GIS: a computing perspective*. CRC press, 2004. 1, 5, 9, 13, 14